ABCs of z/OS System Programming
Volume 10

IBM System z processor design and connectivity

z/Architecture, LPAR concepts, HCD, z196, z10

DS8000 DASD controller

Paul Rogers
Alvaro Salla
Luiz Carlos Orsoni

ibm.com/redbooks
Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

Fifth Edition (March 2012)

This edition applies to Version 1 Release 10 of z/OS (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.
# Contents

**Notices** ........................................................................................................... xi  
**Trademarks** ................................................................................................. xii  

**Preface** ........................................................................................................... xiii  
The team who wrote this book ....................................................................... xiii  

**Chapter 1. Introduction to z/Architecture** ...................................................... 1  
1.1 Computer architecture overview ...................................................... 3  
1.2 Concept of a process .............................................................. 4  
1.3 Process states and attributes ................................................. 5  
1.4 System components ............................................................ 7  
1.5 Processing units (PUs) ......................................................... 10  
1.6 z/Architecture enhancements ............................................... 12  
1.7 64-bit address space map ....................................................... 14  
1.8 Addressing mode ............................................................... 16  
1.9 64-bit dynamic address translation ...................................... 18  
1.10 CP registers (general) ......................................................... 20  
1.11 Floating point registers ...................................................... 22  
1.12 Current program-status word (PSW) .................................... 23  
1.13 Next sequential instruction address ..................................... 24  
1.14 Program-status-word format .............................................. 25  
1.15 Prefixed save area (PSA) ....................................................... 29  
1.16 Several instruction formats ................................................. 30  
1.17 Microcode concepts .......................................................... 32  
1.18 z/Architecture components .................................................. 33  
1.19 z/Architecture data formats ................................................ 36  
1.20 Interrupts ..................................................................... 39  
1.21 Interrupt processing .......................................................... 41  
1.22 Types of interrupts ............................................................ 43  
1.23 Supervisor call interrupt ...................................................... 44  
1.24 Storage protection ............................................................. 48  
1.25 Storage protection logic ....................................................... 50  
1.26 Addresses and address spaces ............................................ 51  
1.27 z/Architecture address sizes ................................................ 53  
1.28 Storage addressing ............................................................. 54  
1.29 Real storage locations ........................................................ 55  
1.30 Dynamic address translation (DAT) ................................. 57  
1.31 Dynamic address translation ............................................... 59  
1.32 Page faults .................................................................. 60  
1.33 Dual address space (cross memory) .................................... 62  
1.34 Access register mode (dataspaces) ...................................... 64  
1.35 CPU signaling facility ....................................................... 66  
1.36 Time measurement TOD .................................................... 67  
1.37 Time measurement (CP timer) ............................................. 69  
1.38 Sysplex Timer expanded availability configuration .............. 71  
1.39 Server Time Protocol (STP) ............................................... 73  
1.40 Data center and I/O configuration ....................................... 74  
1.41 Channel subsystem (CSS) elements .................................. 76  
1.42 Multiple CSS structure (z10 and z196 EC) ......................... 78  

© Copyright IBM Corp. 2008, 2012. All rights reserved.
## Chapter 2. Introducing the IBM z10

2.1 z196 and z10 overview ................................................. 107
2.2 IBM System z nomenclature ........................................... 109
2.3 z10 EC naming summary .............................................. 110
2.4 The power of GHz (high frequency) ................................ 111
2.5 Processor unit (PU) instances ....................................... 113
2.6 z10 EC hardware model .............................................. 116
2.7 z10 EC sub-capacity models ........................................ 117
2.8 z10 EC frames and cages ............................................ 119
2.9 Book topology comparison ......................................... 121
2.10 NUMA topology .................................................... 122
2.11 z10 EC Books ..................................................... 123
2.12 Multi-chip module (MCM) .......................................... 125
2.13 PU chip .......................................................... 126
2.14 Book element interconnections .................................. 128
2.15 Pipeline in z10 EC ................................................ 129
2.16 Pipeline branch prediction ....................................... 131
2.17 About each z10 EC PU ............................................. 132
2.18 z10 EC storage controller (SC) chip ............................... 134
2.19 Recapping the z10 EC design .................................... 136
2.20 Three levels of cache .............................................. 137
2.21 Software/hardware cache optimization .......................... 140
2.22 HiperDispatch considerations ................................... 142
2.23 Central storage design ............................................. 143
2.24 Addresses and addresses ......................................... 144
2.25 Hardware system area (HSA) ...................................... 146
2.26 Large page (1 M) support ....................................... 147
2.27 Connecting PU cage with I/O cages ............................... 149
2.28 Detailed connectivity ............................................. 151
2.29 HCA and I/O card connections ................................ 152
2.30 InfiniBand interconnect technology .............................. 153
2.31 I/O cage ....................................................... 155
2.32 The I/O data flow ............................................... 157
2.33 Redundant I/O Interconnect .................................... 158
2.34 z10 EC I/O features supported ................................ 159
2.35 16-port ESCON channel card .................................... 160
2.36 FICON features and extended distance ......................... 161
2.37 Features in z10 and z196 ......................................... 163
2.38 z10 EC new features .............................................. 164

### Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.43 Control units</td>
<td>80</td>
</tr>
<tr>
<td>1.44 Device number</td>
<td>82</td>
</tr>
<tr>
<td>1.45 Subchannel number</td>
<td>84</td>
</tr>
<tr>
<td>1.46 Subchannel numbering</td>
<td>86</td>
</tr>
<tr>
<td>1.47 Control unit address</td>
<td>88</td>
</tr>
<tr>
<td>1.48 Unit addresses</td>
<td>90</td>
</tr>
<tr>
<td>1.49 Map device number to device address</td>
<td>92</td>
</tr>
<tr>
<td>1.50 Multiple channel paths to a device</td>
<td>93</td>
</tr>
<tr>
<td>1.51 Start subchannel (SSCH) logic</td>
<td>95</td>
</tr>
<tr>
<td>1.52 SAP PU logic</td>
<td>97</td>
</tr>
<tr>
<td>1.53 Channel processing</td>
<td>99</td>
</tr>
<tr>
<td>1.54 I/O interrupt processing</td>
<td>101</td>
</tr>
<tr>
<td>1.55 I/O summary</td>
<td>103</td>
</tr>
</tbody>
</table>
Chapter 3. Introducing the IBM zEnterprise

3.1 zEnterprise overview ................................................................. 167
3.2 Migrating Unix/Linux workloads back to the mainframe .................. 170
3.3 z196 numeric comparison ......................................................... 172
3.4 Processor Units (PU) instances ................................................. 173
3.5 z196 models ............................................................................. 175
3.6 Sub capacity models ................................................................. 176
3.7 Model capacity identifier and MSU/h. ........................................ 178
3.8 z196 frames, cages and I/O drawers (I) ...................................... 179
3.9 z196 frames, cages and I/O drawers (II) ..................................... 180
3.10 NUMA topology ..................................................................... 182
3.11 z196 books ............................................................................. 183
3.12 Any to any book connectivity .................................................. 185
3.13 Fanout cards in a book ............................................................ 186
3.14 Multichip module (MCM) .......................................................... 188
3.15 Frequency (GHz) in a z196 PU ................................................ 189
3.16 Quad core PU chip .................................................................. 191
3.17 PU chip coprocessor ............................................................... 193
3.18 Storage controller (SC) chip in MCM ......................................... 197
3.19 z196 book recapping ............................................................... 199
3.20 Pipeline concept within a PU .................................................. 200
3.21 Out of order execution ............................................................ 202
3.22 z196 instructions ................................................................... 203
3.23 Non-quesce SSKE instruction .................................................. 205
3.24 z10 EC and z196 cache design comparison ................................. 207
3.25 Storage layers ....................................................................... 208
3.26 z196 cache design (I) ............................................................... 210
3.27 z196 cache design (II) ............................................................. 212
3.28 HiperDispatch concepts .......................................................... 215
3.29 The clerk dilemma .................................................................. 217
3.30 z/OS dispatcher logic in HiperDispatch ..................................... 219
3.31 Central Storage design ............................................................. 220
3.32 MCUs and DIMMs in a z196 book ............................................ 221
3.33 Purchase memory offerings ..................................................... 222
3.34 Addresses and addresses ......................................................... 223
3.35 Hardware system area (HSA) .................................................... 225
3.36 Large pages ............................................................................ 226
3.37 An I/O data flow tree analogy ................................................ 228
3.38 Book to channel connectivity ................................................... 229
3.39 Connecting books with I/O channels ....................................... 230
3.40 FICON I/O card ..................................................................... 231
3.41 FICON channel topics in z196 .................................................. 232
3.42 I/O cage ................................................................................. 234
3.43 I/O drawer .............................................................................. 235
3.44 Redundant I/O interconnect ..................................................... 236
3.45 Coupling Facility links ............................................................. 237
3.46 Infiniband protocol ................................................................. 238
3.47 z196 maximum number of channel per type ............................... 240
3.48 z/OS discovery and auto-configuration (zDAC) ......................... 242
3.49 WWPN and fabrics discovery ................................................... 243
3.50 zDAC software and hardware requirements ............................... 244
3.51 zDAC policy in HCD ............................................................... 245
3.52 zDAC policy ......................................................................... 246
Contents

8.1 What is HCD ................................................. 492
8.2 IOCP example .................................................. 494
8.3 IOCP elements ................................................. 495
8.4 Hardware and software configuration ......................... 497
8.5 HCD functions ................................................ 499
8.6 Dynamic I/O reconfiguration .................................. 501
8.7 Dynamic I/O reconfiguration device types ..................... 503
8.8 IODF data set ................................................. 505
8.9 Definition order ............................................... 508
8.10 HCD primary menu .......................................... 509
8.11 Creating a new work IODF .................................... 510
8.12 Defining configuration data ................................... 511
8.13 Operating system definition .................................. 512
8.14 Defining an operating system ................................ 513
8.15 EDT and esoterics ............................................ 514
8.16 How to define an EDT (1) .................................... 516
8.17 How to define an EDT (2) .................................... 517
8.18 Defining an EDT identifier .................................... 518
8.19 How to add an esoteric ...................................... 519
8.20 Adding an esoteric ........................................... 521
8.21 Defining switches ............................................. 522
8.22 Adding switches .............................................. 524
8.23 Defining servers .............................................. 525
8.24 Information for defining a server .............................. 527
8.25 Defining a server ............................................. 528
8.26 Working with LCSSs .......................................... 529
8.27 Logical channel subsystems defined ......................... 530
8.28 Adding a logical partition (LP) ............................... 531
8.29 z9 EC LPAR server configuration ............................ 532
8.30 Channel operation mode ...................................... 533
8.31 Channel types ................................................ 534
8.32 Information required to add channels ....................... 536
8.33 Working with channel paths .................................. 537
8.34 Adding channel paths dynamically ......................... 538
8.35 Adding a channel path ....................................... 539
8.36 Defining an access and a candidate list ..................... 540
8.37 Adding a control unit ....................................... 541
8.38 Information required to define a control unit ................ 542
8.39 Adding a control unit ....................................... 543
8.40 Defining a 2105 control unit ................................ 544
8.41 Selecting a processor/control unit ............................ 545
8.42 Servers and channels for connecting control units .......... 547
8.43 Defining server attachment data .............................. 548
8.44 Information required to define a device .................... 549
8.45 z/OS device numbering ....................................... 551
8.46 Defining a device ............................................ 552
8.47 Defining device CSS features (1) ............................ 556
8.48 Defining device CSS features (II) ............................ 557
8.49 Defining devices to the operating system ...................... 558
8.50 Defining operating system device parameters ............... 559
8.51 Assigning a device to an esoteric ........................... 561
8.52 Defining an NIP console ..................................... 562
8.53 Using the CHPID mapping tool ............................... 563
Chapter 9. **DS8000 series concepts**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 DASD controller capabilities</td>
<td>578</td>
</tr>
<tr>
<td>9.2 DS8000 characteristics</td>
<td>580</td>
</tr>
<tr>
<td>9.3 DS8000 design</td>
<td>582</td>
</tr>
<tr>
<td>9.4 Internal fabric and I/O enclosures</td>
<td>583</td>
</tr>
<tr>
<td>9.5 Disk subsystem</td>
<td>584</td>
</tr>
<tr>
<td>9.6 Switched Fibre Channel Arbitrated Loop (FC-AL)</td>
<td>586</td>
</tr>
<tr>
<td>9.7 Redundant array of independent disks (RAID)</td>
<td>588</td>
</tr>
<tr>
<td>9.8 DS8000 types of RAID</td>
<td>590</td>
</tr>
<tr>
<td>9.9 Logical subsystem (LSS)</td>
<td>592</td>
</tr>
<tr>
<td>9.10 Logical partition (LPAR)</td>
<td>593</td>
</tr>
<tr>
<td>9.11 Copy services classification criteria</td>
<td>594</td>
</tr>
<tr>
<td>9.12 Consistency group concept</td>
<td>596</td>
</tr>
<tr>
<td>9.13 Copy services in DS8000</td>
<td>598</td>
</tr>
<tr>
<td>9.14 FlashCopy</td>
<td>599</td>
</tr>
<tr>
<td>9.15 Consistency group in FlashCopy</td>
<td>602</td>
</tr>
<tr>
<td>9.16 Remote Mirror and Copy (example: PPRC)</td>
<td>603</td>
</tr>
<tr>
<td>9.17 Consistency groups in Metro Mirror</td>
<td>605</td>
</tr>
<tr>
<td>9.18 Global Copy (example: PPRC XD)</td>
<td>607</td>
</tr>
<tr>
<td>9.19 Global Mirror (example: async PPRC)</td>
<td>608</td>
</tr>
<tr>
<td>9.20 z/OS Global Mirror (example: XRC)</td>
<td>610</td>
</tr>
<tr>
<td>9.21 Parallel access volume (PAV)</td>
<td>612</td>
</tr>
<tr>
<td>9.22 HyperPAV feature for DS8000 series</td>
<td>614</td>
</tr>
<tr>
<td>9.23 HyperPAV and IOS</td>
<td>616</td>
</tr>
<tr>
<td>9.24 HyperPAV implementation</td>
<td>617</td>
</tr>
<tr>
<td>9.25 Display M=DEV command</td>
<td>619</td>
</tr>
<tr>
<td>9.26 RMF DASD report</td>
<td>620</td>
</tr>
<tr>
<td>9.27 RMF I/O Queueing report</td>
<td>621</td>
</tr>
<tr>
<td>9.28 DS8000 Capacity on Demand</td>
<td>622</td>
</tr>
<tr>
<td>9.29 DS command line interface (CLI)</td>
<td>623</td>
</tr>
<tr>
<td>9.30 Storage Hardware Management Console (S-HMC)</td>
<td>624</td>
</tr>
</tbody>
</table>

**Related publications**

- IBM Redbooks ............................................................................................................. 625
- Other publications ........................................................................................................ 625
- Online resources .......................................................................................................... 625
- How to get IBM Redbooks ......................................................................................... 626
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

- AIX®
- BladeCenter®
- CICS®
- Cool Blue®
- DataPower®
- DB2®
- DS6000™
- DS8000®
- ECKD™
- Enterprise Storage Server®
- ESCON®
- FICON®
- FlashCopy®
- GDPS®
- Geographically Dispersed Parallel Sysplex™
- HiperSockets™
- HyperSwap®
- i5/OS®
- IBM Systems Director Active Energy Manager™
- IMS™
- Language Environment®
- MQSeries®
- MVS™
- Orchestrator®
- Parallel Sysplex®
- Power Systems™
- POWER6®
- POWER7®
- PowerPC®
- PowerVM®
- PR/SM™
- Processor Resource/Systems Manager™
- pSeries®
- RACF®
- Redbooks®
- Redbooks (logo) ®
- Resource Link®
- RMF™
- S/390®
- Sysplex Timer®
- System p®
- System Storage®
- System x®
- System z10®
- System z9®
- System z®
- VTAM®
- WebSphere®
- z/Architecture®
- z/OS®
- z/VSE®
- z10™
- z9®
- zEnterprise™
- zSeries®

The following terms are trademarks of other companies:

- Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

- Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

- Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

- UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.
Preface

The ABCs of IBM® z/OS® System Programming is an 13-volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you would like to become more familiar with z/OS in your current environment, or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection will serve as a powerful technical tool.

This IBM Redbooks® publication, Volume 10, provides an introduction to IBM z/Architecture®, IBM zSeries® processor design, zSeries connectivity, LPAR concepts, Hardware Management Console (HMC), Hardware Configuration Definition (HCD), and IBM DS8000®.

The contents of the other volumes are as follows:

- Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation
- Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLST, authorized libraries, SMP/E, IBM Language Environment®
- Volume 3: Introduction to DFSMS, data set basics storage management hardware and software, catalogs, and DFSMSstvs
- Volume 4: Communication Server, TCP/IP, and IBM VTAM®
- Volume 5: Base and IBM Parallel Sysplex®, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), IBM Geographically Dispersed Parallel Sysplex™ (IBM GDPS®)
- Volume 6: Introduction to security, IBM RACF®, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries firewall technologies, LDAP, and Enterprise Identity Mapping (EIM)
- Volume 7: Printing in a z/OS environment, Infoprint Server and Infoprint Central
- Volume 8: An introduction to z/OS problem diagnosis
- Volume 9: z/OS UNIX System Services
- Volume 10: Introduction to z/Architecture, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and DS8000
- Volume 11: Capacity planning, performance management, WLM, IBM RMF™, and SMF
- Volume 12: WLM
- Volume 13: JES3, JES3 SDSF

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Paul Rogers is a Consulting IT Specialist at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes.
worldwide on various aspects of z/OS, z/OS UNIX, and Infoprint Server. Before joining the ITSO 23 1/2 years ago, he worked in the IBM Installation Support Center (ISC) in Greenford, England providing JES support for IBM EMEA and the Washington Systems Center. Paul has worked for IBM for 44 1/2 years.

**Alvaro Salla** is an IBM retiree who worked for IBM for more than 30 years, specializing in large systems. He has co-authored many IBM Redbooks publications and spent many years teaching about large systems from S/360 to IBM S/390®. He has a degree in Chemical Engineering from the University of Sao Paulo, Brazil.

**Luiz Carlos Orsoni** is a Technical Sales Support member of the Linux for zSeries team in Brazil. He has four years of experience in the areas of operational systems and networking. He is finishing college and has been working for IBM since 2002, with responsibility for planning and implementing new workload projects. He has been a contributor to other IBM Redbooks publications.

Thanks also to Robert Haimowitz, International Technical Support Organization, Poughkeepsie Center, for his support of this project.

**Now you can become a published author, too!**

Here’s an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at: [ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)
Comments welcome

Your comments are important to us!
We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

Use the online Contact us review Redbooks form found at:

ibm.com/redbooks

Send your comments in an email to:

redbooks@us.ibm.com

Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

Find us on Facebook:

http://www.facebook.com/IBMRedbooks

Follow us on Twitter:

http://twitter.com/ibmredbooks

Look for us on LinkedIn:

http://www.linkedin.com/groups?home=&gid=2130806

Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:


Stay current on recent Redbooks publications with RSS Feeds:

http://www.redbooks.ibm.com/rss.html
Introduction to z/Architecture

This chapter provides an overview of z/Architecture.

z/Architecture is the next step in the evolution from System/360 to System/370, System/370 Extended Architecture (370-XA), Enterprise Systems Architecture/370 (ESA/370), and Enterprise Systems Architecture/390 (ESA/390). In order to understand z/Architecture, you have to be familiar with the basics of ESA/390 and its predecessors.

An address space maps all of the available addresses—and includes system code and data, and may include user code and data. Thus, not all of the mapped addresses are available for user code and data. This limit on user applications was a major reason for System/370 Extended Architecture (370-XA) and MVS/XA. Because the effective length of an address field expanded from 24 bits to 31 bits, the size of an address space expanded from 16 megabytes to 2 gigabytes. An MVS/XA address space is 128 times as big as an MVS/370 address space. Similarly, when the address field grew from 31 to 64 bits in zArchitecture, the maximum possible size of the Address Space went from 2 gigabytes to 16 exabytes, 8 billion times as big as a “normal”, 31 bit, 2 gigabytes address space.

In the early 1980s, XA (or eXtended Architecture) was introduced with an address space that began at address 0 and ended at two gigabytes. The architecture that created this address space provided 31-bit addresses. To maintain compatibility, IBM MVS™ provided two addressing modes (AMODEs) for programs: programs that run in AMODE 24 can use only the first 16 megabytes of the address space, and programs that run in AMODE 31 can use the entire 2 gigabytes.

As of z/OS V1R2, the address space can begin at address 0 and end at 16 exabytes, an incomprehensibly high address. This architecture, zArchitecture, in creating this address space, provides 64-bit addresses and AMODE 64 to allow access to data above the bar at 4 gigabytes (programs still cannot be executed above the bar). The address space structure below the 2 gigabyte address has not changed; all programs in AMODE 24 and AMODE 31 continue to run without change. In some fundamental ways, the address space is much the same as the XA address space.

z/OS and the IBM zSeries deliver the 64-bit architecture (z/Architecture) to provide qualities of service that are critical for the e-business world. 64-bit real storage support eliminates expanded storage and helps eliminate paging. 64-bit real storage support may allow to
consolidate your current systems into fewer logical partitions (LPARs), or to a single native image.

With System z®, initial program loading sets the ESA/390 architectural mode. The new SIGNAL PROCESSOR order then can be used to set the z/Architecture mode or to return from z/Architecture to ESA/390. This order causes all CPs in the configuration to always be in the same architectural mode.

When converting to z/Architecture mode, existing programs work unchanged.

In this book, the following terminology is used to refer to the operating system:

- When appropriate, we specifically emphasize z/OS.
- Otherwise, if a statement is valid for any previous release or version of z/OS, we may use MVS in the statement.
1.1 Computer architecture overview

This chapter provides a detailed description of z/Architecture, and it discusses aspects of the servers running such architecture. These servers are now referred to as System z and they comprise the z800, z890, z900, z990, z9® EC and BC, z10™ EC and BC, z196 EC and BC (zEnterprise™) and zBX (Blade Center).

**Note:** When reading this chapter, it is useful to have z/Architecture Reference Summary, SA22-7871, at hand to use as a quick reference. The main source of reference information about this architecture, however, is z/Architecture Principles of Operations, SA22-7832.

So, what is a “computer architecture”? The computer architecture of a computing system defines its attributes as seen by the programs that are executed in that system, that is, the conceptual structure and functional behavior of the server hardware. Then, the computer architect defines the functions to be executed in the hardware and the protocol to be used by the software in order to exploit such functions. Note that the architecture has nothing to do with the organization of the data flow, the logical design, the physical design, and the performance of any particular implementation in the hardware.

Several dissimilar server implementations may conform to a single architecture. When the execution of a set of programs on different server implementations produces the results that are defined by a single architecture, the implementations are considered to be compatible for those programs.
1.2 Concept of a process

- Real Life
  - Car
  - Street
  - Going to the movies

- Data Processing
  - Program
  - Process
  - CP

Figure 1-2  Concept of a process

Architecture concepts

One of the z/Architecture items we cover in this document is multiprocessing. However, in order to understand multiprocessing, one first need to be familiar with the concept of a process. As defined in a commercial data processing environment, a process is the serial execution of programs in order to solve one problem of a business unit (such as payroll update, banking checking account transaction, Internet query, and so on).

Analogies

To illustrate the process concept, let’s draw an analogy using real life. Each item in the real-life list in Figure 1-2 corresponds to an item in the data processing list: the car is the CP, the street is the program, and “going to the movies” is the process. Streets and cars were created because people go to places. Programs and CPs (and thus, the data processing industry) were developed because processes need to be executed. So, if every time you go to the movies, you notice that you are late because you have a slow car, it would be useless to buy another slow car—in the same way, if your process is taking longer because of a slow CP, it is useless to add another server that has the same speed. You could also go to the movies by taking a taxi, then get out somewhere, do some shopping, take another taxi, and so on. Similarly, the same process can start in one CP, be interrupted, and later resume in another CP, and so on. Also, on the same street there may be different cars taking people to different places. Likewise, the same reentrant program can be executed by different CPs on behalf of different processes. Finally, taking someone to the hospital allows you to shortcut a queue in the traffic, just as a key process has a higher priority in a queue for getting CP and I/O compared with other processes.
1.3 Process states and attributes

- **Process states**
  - **Active**, when its program is executing on a CP
  - **Ready**, when being delayed due to CP availability
  - **Wait**, when delayed by any reason other than a CP

- **Process attributes**
  - State
  - Resources
  - Priority
  - Accounting
  - Addressing
  - Security

States of processing
From an operating system perspective, a process has one of three states:

- **Active**  In the Active state, a program is being executed by one CP.
- **Ready**  In the Ready state, a program is delayed because all available CPs are busy executing other processes.
- **Wait**  In the Wait state, a program is being delayed for a reason that is not CP-related (for example, waiting for an event, such as an I/O operation to complete).

Process attributes
In the operating system, processes are born and die (normally or abnormally), depending on the needs of business units. A process dies *normally* when its last program completes (i.e., its last instruction gets executed) normally. A process dies *abnormally* when one of its programs (in fact, one of its instructions) tries to execute something wrong, or forbidden. The amount of resources consumed is charged to the process, and not to the program. Also, when there are queues for accessing resources, the priority to be placed in such queues depends on the process and not on the program.

For each active process in the system, z/OS must keep the following:
- State (PSW, content of registers)
- Resources held (data sets, virtual storage, programs in memory, ENQs)
Priority when entering in queues for resources
Accounting - how much CP time was used, I/Os executed, and memory occupied
Addressing - which addresses the programs of the process have the right to access in memory (for data and instructions)
Security profile - which resources are allowed to the process

Dispatchable units
In z/OS, processes are called DUs - dispatchable units, which consist of tasks and service requests. The control program creates a task in the address space, as a result of initiating execution of the process (called the job step task), by means of an ATTACH Macro (SVC 42).

One can create additional tasks in a program. However, if you do not, the job step task is the only active task in the address space being executed. The benefits of a multiprogramming and multiprocessing environment are still available even with only one task in the job step; work is still being performed for other address spaces when your task is waiting for an event, such as an input operation, to occur.

Note: From now on in this publication, we will use the word “task” instead of “process”.
1.4 System components

Before giving more detailed information about z/Architecture, we will describe a system generically.

Physically, a system consists of the following:

- Main storage.
- One or more central processing units—previously known as CPU, but in this publication we use the term central processor (CP).
- Operator facilities (Service Element, which is not represented in Figure 1-4).
- A channel subsystem (formed by system assisted processors (SAPs) and channels).
- I/O devices (for example, disks also called DASD, tape, printers, teleprocessing); I/O devices are attached to the channel subsystem through control units. The connection between the channel subsystem and a control unit is called a channel path.

Figure 1-4  System components
System Assist Processor (SAP)
System Assist Processor (SAP) is, physically, exactly the same as a CP, but with a different microcode; refer to “Microcode concepts” on page 32 for more information. The SAP acts as an offload engine for the CPs. Different server models have different numbers of SAPs. The SAP relieves CP involvement, thus guaranteeing one available channel path to execute the I/O operation. In other words, it schedules and queues an I/O operation, but it is not in charge of the movement between central storage (CS) and the channel.

Channels
A channel is much simpler in that it assists in the dialog with an I/O control unit, to execute an I/O operation—that is, the data transfer from or to memory and the device.

Previously, there was no need for channels because only one process at a time (for example, a payroll) was loaded in storage. So if this process needed an I/O operation, the CP itself executed it—that is, it communicated with the I/O control unit. There was no other process to be executed in memory.

However, now that we are able to have several processes in memory at the same time (doing multiprogramming and multiprocessing), using the CP to entertain the I/O operations is inefficient. The CP is an expensive piece of hardware, and other independent processes may require processing. For this reason, the concept of using SAPs and channels was introduced.

Channel paths
A channel path employs either a parallel-transmission electric protocol (old fashion) or a serial-transmission light protocol and, accordingly, is called either a parallel channel path or a serial channel path. For better connectivity and flexibility, a serial channel may connect to a control unit through a dynamic switch that is capable of providing multiple connections between entities connected to the ports of the switch (that is, between channels and I/O control units).

Expanded storage
Expanded storage is a sort of second level memory introduced because of the architected limitation of the 2 GB size of central storage per MVS image. It is not available in z/Architecture, where this 2 GB limitation does not exist anymore. Even though not directly supported by the Hardware, Expanded Storage can be defined for use with z/VM® guests and can be simulated by z/OS.

Crypto
To speed up cryptographic computing, a cryptographic facility can be accessed by a CP. Inside the PU chip, each two CPs share a crypto facility and its cache. The IBM common cryptographic architecture (CCA) defines a set of cryptographic functions, external interfaces, and a set of key management rules which pertain to both the Data Encryption Standard.
(DES)-based symmetric algorithms and the Public Key Algorithm (PKA) asymmetric algorithms.

**ETR**

An external time reference (ETR) may be connected to the server to guarantee time synchronization between distinct servers. The optional ETR cards provide the interface to IBM Sysplex Timers, which are used for timing synchronization between systems in a sysplex environment.
1.5 Processing units (PUs)

Processing units
The following types of processing units (PUs) can be ordered (enabled or assigned) on a System z:

CPU
A CPU is a general purpose processor that is able to execute all the possible z10 and z196 EC running operating systems, such as z/OS, Linux, z/VM, z/VSE®, Coupling Facility Control Code (CFCC), and z/TPF. A CPU is also known as a CP. All other PU types are less expensive than CPUs.

IFL
This type of PU is only able to execute native Linux and Linux under z/VM. IFLs are less expensive than CPUs.

ICF
This type of PU is only able to execute the CFCC operating system. The CFCC is loaded in a Coupling Facility LP from a copy in HSA; after this, the LP is activated and IPLed. ICFs are less expensive than CPUs.

zAAP
This type of PU only runs under z/OS, and is for the exclusive use of Java interpreter code (JVM) and IBM DB2® 9 XML parsing workloads. A zAAP is less expensive than a CPU, and does not increase the software price (based on produced MSUs) because it does not produce MSUs.

zIIP
This type of PU is to run in z/OS only, for eligible DB2 workloads such as DDF, business intelligence (BI), ERP, CRM and IPSec (an open networking function used to create highly secure crypto connections between two points in an enterprise) workloads. A zIIP is less expensive than a CPU and does not increase the software price (based on produced MSUs) because it does not
produce MSUs. There is a limitation to using zIIP processors, however: only a percentage of the candidate workload can be executed on them.

**SAP**  
A System Assist Processor (SAP) is a PU that runs the Channel Subsystem Licensed Internal Code. An SAP manages the starting of I/O operations required by operating systems running in all logical partitions. It frees z/OS (and the CPU) from this role, and is “mainframe-unique”. z10 and z196 EC models have a variable number of standard SAPs configured.

**Spare**  
A spare PU is a PU that is able to replace, automatically and transparently, any falling PU in the same book, or in a different book. There are at least two spares per z10 and z196 EC server.
1.6 z/Architecture enhancements

- 64-bit address space map
- 64- or 31- or 24-bit virtual storage addressing modes
- Introduction of up to 3 levels of region tables
- 64-bit registers
- PSW with 16 bytes (128 bits)
- PSA with 8-KB
- SIGP instruction to introduce a bi-modal architecture
- New instructions

**z/Architecture**

Figure 1-6 lists all the enhancements introduced by z/Architecture in relation to the former ESA/390. The majority of the enhancements are implemented to support 64-bit addressing mode.

With z/OS, the MVS address space expands to a size so vast that we need new terms to describe it. Each address space, called a 64-bit address space, is 16 exabytes in size (an exabyte is slightly more than ten billion gigabytes, see the exact value below).

The new address space has logically 2\(^{64}\) addresses. It is 8 billion times the size of the former 2 gigabyte address space, that logically has 2\(^{31}\) addresses. This number is 18 with 18 places after it: 18,446,744,073,709,551,616 bytes, or 16 exabytes.

**Set addressing mode instructions**

z/Architecture provides three new set addressing mode instructions that allow you to change addressing mode. The instructions are:

- SAM24, which changes the current AMODE to 24
- SAM31, which changes the current AMODE to 31
- SAM64, which changes the current AMODE to 64

The addressing mode also determines where the storage operands can reside. The storage operands for programs running in AMODE 64 can be anywhere in the 16-exabyte address space.
space, while a program running in AMODE 24 can use only storage operands that reside in the first 16 megabytes of the 2 gigabyte, or 16-exabyte, address space.

**z/Architecture enhancements**

z/Architecture also provides significant extensions, as follows:

- Up to three additional levels of dynamic address-translation (DAT) tables, called *region tables*, for translating 64-bit virtual addresses. A virtual address space may be specified either by a segment-table designation as in ESA/390, or by a region-table designation.

- 64-bit general registers and control registers. The bit positions of the general registers and control registers of z/Architecture are numbered 0-63. To maintain compatibility, an ESA/390 instruction that operates on bit positions 0-31 of a 32-bit register in ESA/390, operates instead on bit positions 32-63 of a 64-bit register in z/Architecture.

- An 8K-byte prefix area for containing larger old and new PSWs and register save areas.

- A SIGNAL PROCESSOR order for switching between the ESA/390 and z/Architecture architectural modes. Initial program loading sets the ESA/390 architectural mode. The new SIGNAL PROCESSOR order then can be used to set the z/Architecture mode or to return from z/Architecture to ESA/390. This order causes all CPUs in the configuration always to be in the same architectural mode.

- Many new instructions, many of which operate on 64-bit binary integers and three types of instructions, new since the z990s, able to declare a displacement of 20 bits (1 MB) instead of 12 bits (4 KB), such as RSY, RXY and SIY.
1.7 64-bit address space map

64-bit virtual address space

The virtual storage 2 GB limit was broken with z/Architecture, as explained here.

Support for up to 4 TB of real memory on a single z/OS image (z/OS V1R8). This will allow for up to 1 TB of real memory on a single z/OS image for the z10 and z196 EC server, up to 512 GB of real memory on a single z/OS image on IBM System z9® servers, and up to 256 GB on z990 servers.

Because of changes in the architecture that supports the MVS operating system, there were two different address spaces prior to the 64-bit address space. The address space of the 1970s began at address 0 and ended at 16 megabytes (shown as the 16M line in Figure 1-7). The architecture that created this address space provided 24-bit addressing.

The initial support for 64-bit virtual addressing was introduced in z/OS Version 1 Release 2. The size of the 64-bit address space is 16 exabytes (16 E), which makes the new address space 8 billion times the size of the former S/390 address space. Programs continue to be loaded and run below the 2 gigabyte address; these programs can use data that resides above 4 gigabytes.

Each address space can be logically 16 exabytes in size. To allocate and release virtual storage above 2 GB, a program must use the services provided in the IARV64 macro. The GETMAIN, FREEMAN, STORAGE, and CPOOL macros do not allocate storage above the 4 gigabyte address, nor do callable cell pool services.
The bar

For compatibility, the layout of the storage areas for an address space is the same below 2 GB, providing an environment that can support both 24-bit and 31-bit addressing. The area that separates the virtual storage area below the 2 GB address from the user private area is called the bar, as shown in Figure 1-7 on page 14.

In a 64-bit virtual storage environment, the terms “above the bar” and “below the bar” are used to identify the areas between $2^{32}$ and $2^{31}-1$, and 0 and $2^{31}-1$, respectively. For example, any address in the range 0 to $7FFFFFFF$ is below the bar, and an address in the range $00000000_00000000$ to $7FFFFFFFFFFF$ is above the bar. This is basically an addition to the 2 GB 31-bit terminology that related “below the line” to 24-bit storage, and “above the line” to 31-bit addresses.

The 64-bit address space map is as follows:

- **0 to $2^{31}-1$:** The layout is the same as in all the previous releases that supported 31-bit addressing. Unless explicitly asked for, address spaces continues to have 2 gigabytes.

- **$2^{31}$ to $2^{32}-1$:** From 2 GB to 4 GB is considered the bar. Below the bar can be addressed with a 31-bit address. Above the bar requires a 64-bit address. Just as the system does not back the page at $7FFFF000$ in order to protect programs from addresses which can wrap to 0, the system does not back the virtual area between 2 GB and 4 GB. That means a 31-bit address with the high bit on will always program check if used in AMODE 64.

- **$2^{32}$ - $2^{41}-1$:** The Low Non-shared area starts at 4G and goes to $2^{41}-1$.

- **$2^{41}$ - $2^{50}-1$:** The Shared Area starts at $2^{41}$ and goes to $2^{50}-1$ or higher if requested.

- **$2^{50}$ - $2^{64}-1$:** The High Non-shared area starts at $2^{50}$ or wherever the Shared Area ends and goes to $2^{64}-1$.

**Note:** The Shared Area is supported beginning with z/OS Version 1 Release 5.

User private area

The area above the bar is intended for application data; no programs run above the bar. No system information or system control blocks exist above the bar, either. Currently there is no common area above the bar. However, IBM reserves an area above the bar to be used for future enhancements. A user program can also reserve some area in the virtual storage allocated above the bar.

The user private area includes:

- Low private: The private area below the line
- Extended private: The private area above the line
- Low Non-shared: The private area just above the bar
- High Non-shared: The private area above Shared Area

As users allocate private storage above the bar, it will first be allocated from the Low Non-shared area. Similarly, as Shared Area is allocated, it will be allocated from the bottom up. This is done to allow applications to have both private and shared memory above the bar and avoid additional server cycles to perform dynamic address translation (DAT).
1.8 Addressing mode

A program module has a residence mode assigned to it, and each entry point and alias has an addressing mode assigned to it. You can specify one or both of these modes when creating a program module, or you can allow the binder to assign default values. AMODEs and RMODEs can be assigned at assembly or compilation time for inclusion in an object module.

One assign an addressing mode (AMODE) to indicate which hardware addressing mode is active when the program executes. Addressing modes are:

- **24**: Indicates that 24-bit addressing must be in effect.
- **31**: Indicates that 31-bit addressing must be in effect.
- **ANY**: Indicates that either 24-bit or 31-bit addressing can be in effect.
- **64**: Indicates that 64-bit addressing can be in effect.

Note: Even though ANY is still used, it restricts to only 24-bit and 31-bit addressing modes and does NOT includes 64 bit addressing mode!

Running programs

The addressing mode determines where storage operands can reside. The storage operands for programs running in AMODE 64 can be anywhere in the 16-exabyte address space, while
a program running in AMODE 24 can use only storage operands that reside in the first 16 megabytes of the 16-exabyte address space.

When generating addresses, the server performs address arithmetic; it adds three components: the contents of the 64-bit GPR, the displacement (a 12-bit value), and (optionally) the contents of the 64-bit index register.

Then, the server checks the addressing mode and truncates the answer accordingly. For AMODE 24, the server truncates bits 0 through 39; for AMODE 31, the server truncates bits 0 through 32; for AMODE 64, no truncation (or truncation of 0 bits) occurs.

An AMODE value is provided for each entry point into the program module. The main program AMODE value is stored in the primary directory entry for the program module. Each alias directory entry contains the AMODE value for both the main entry point and the alias or alternate entry point.

When a program is loaded in memory, its addressing mode is already determined. There are non-privileged instructions that are able to change the addressing mode, such as BRANCH AND SET MODE (BSM) and BRANCH AND SAVE AND SET MODE (BASSM).

**Architecture considerations**

In S/390 architecture, the processor added together the contents of a 32-bit GPR, the 12 bit displacement, and (optionally) the contents of a 32-bit index register. It then checked to see if the addressing mode was 31 or 24 bits, and truncated accordingly. AMODE 24 caused truncation of 8 bits; AMODE 31 caused a truncation of bit 0.

For total compatibility with old code, z/Architecture handles three possibilities:

- 24 bits (AMODE 24), up to 16 MB addresses
- 31 bits (AMODE 31), up to 2 GB addresses
- 64 bits (AMODE 64), up to 16 EB addresses

The CP addressing mode for the running program is described in bits 31 and 32 in the current PSW, respectively:

- 00 indicates AMODE 24.
- 01 indicates AMODE 31.
- 10 is **invalid**.
- 11 indicates AMODE 64.
1.9 64-bit dynamic address translation

Region tables for 64-bit

In a 16-EB address space with 64-bit virtual storage addressing, there are three additional levels of translation tables, called region tables. They are called region third table (R3T), region second table (R2T), and region first table (R1T). The region tables are 16 KB in length, and there are 2048 entries per table. Each region has 2 GB.

When the first storage is created above the bar, RSM creates the R3T. The R3T table has 2048 segment table pointers, and provides addressability to 4 TB. When virtual storage greater than 4 TB is allocated, an R2T is created. An R2T has 2048 R3T table pointers and provides addressability to 8 PB. An R1T is created when virtual storage greater than 8 PB is allocated. The R1T has 2048 R2T table pointers, and provides addressability to 16 EB. Figure 1-9 shows the page table hierarchy and the size of virtual storage each table covers.

Segment tables and page table formats remain the same as for virtual addresses below the bar. When translating a 64-bit virtual address, once you have identified the corresponding 2G region entry that points to the segment table, the process is the same as that described previously.

RSM only creates the additional levels of region tables when necessary to back storage that is mapped. They are not built until a translation exception occurs. So for example, if an application requests 60 PB of virtual storage, the necessary R2T, R3T, segment table, and page tables are only created if they are needed to back a referenced page. Up to five lookup
tables may be needed by DAT to do translation, but the translation only starts from the table that provides translation for the highest usable virtual address in the address space.
1.10 CP registers (general)

General registers are used to hold temporary data:
- Machine instructions loaded from memory
- Fixed point data also called binary integers

Registers
The CP provides registers that are available to programs, but that do not have addressable representations in main storage. They include the current program-status word (PSW), the general registers, the floating-point registers and floating-point-control register, the control registers, the access registers, the prefix register, and the registers for the clock comparator and the CP timer.

Each CP in an installation provides access to a time-of-day (TOD) clock, which is shared by all CPs in the installation. The instruction operation code determines which type of register is to be used in an operation. There are several types of registers, as explained in the following sections.

General registers
General registers (GRs) are used to keep temporary data (operands) loaded from memory to be processed or already processed. Instructions may designate information in one or more of 16 general registers. The general registers may be used as base-address registers and index registers in address arithmetic, and as accumulators in general arithmetic and logical operations.

Each register contains 64 bit positions. The general registers are identified by the numbers 0-15, and are designated by a four-bit R field in an instruction. Some instructions provide for addressing multiple general registers by having several R fields. For some instructions, the use of a specific general register is implied rather than explicitly designated by an R field of
the instruction. The data is in binary integer format, also called fixed point. There are certain CP instructions that are able to process data stored in GRs. Its contents can also be used for the execution of a CP instruction to point to the address of a storage operand.

When an Instruction needs only 32 bits, such as Load, Add and Store, only bits 32 to 63 of the General Register are used, the high order bits 0 to 31 are not disturbed. Of course, there are Instructions such as LG, AG and STG that handles all 64 bits of a General Register.

Control registers
The CP has 16 control registers (CRs), each having 64 bit positions. The bit positions of some of the control registers are assigned to particular facilities in the system, such as program-event recording, and are used either to specify that an operation can take place, or to furnish special information required by the facility.

The control registers are identified by the numbers 0-15 and are designated by four-bit R fields in the instructions LOAD CONTROL and STORE CONTROL. Multiple control registers can be addressed by these instructions.

CRs are registers accessed and modified by z/OS through privileged instructions. All the data contained in the CRs are architected containing information input by z/OS and used by hardware functions (such as Crypto, cross memory, virtual storage, and clocks) implemented in the server. It is a kind of extension of the PSW (covered in “Current program-status word (PSW)” on page 23). Refer to z/Architecture Reference Summary, SA22-7871, for the complete set of CR contents.

Access registers
Access registers (ARs) are used by z/OS to access data spaces and other address spaces, through activating the access register mode in the CP; this subject is covered in more detail in “Access register mode (dataspaces)” on page 64.

The CP has 16 access registers numbered 0-15. An access register consists of 32 bit positions containing an indirect specification of an address-space-control element. An address-space-control element is a parameter used by the dynamic-address-translation (DAT) mechanism to translate references to a corresponding data space or address space.

When the CP is in a mode called the access-register mode (controlled by bits 16 and 17 in the PSW), an instruction B field, used to specify a logical address for a storage-operand reference, designates an access register, and the address-space-control element specified by the access register is used by DAT for the reference being made. For some instructions, an R field is used instead of a B field. Instructions are provided for loading and storing the contents of the access registers, and for moving the contents of one access register to another.
Floating point registers

All floating-point instructions (HFP, BFP, and DFP) use the same floating-point registers. The CP has 16 floating-point registers. The floating-point registers are identified by the numbers 0-15 and are designated by a four-bit R field in floating-point instructions. Each floating-point register is 64 bits long and can contain either a short (32-bit) or a long (64-bit) floating-point operand. As shown in Figure 1-11, pairs of floating-point registers can be used for extended (128-bit) operands. When using extended operand instructions, each of the eight pairs is referred to by the number of the lower-numbered register of the pair.

Floating point registers (FPRs) are used to keep temporary data (operands) loaded from memory to be processed or already processed. This data is in the format HFP, BFP or DFP.

There is also a floating-point-control (FPC) register, a 32-bit register to control the floating point instructions execution. It contains mask bits, flag bits, a data exception code, and rounding-mode bits.
1.12 Current program-status word (PSW)

Figure 1-12  PSW from bit 0 to bit 31

Program-status word (PSW)
The current PSW is a storage circuit located within the CP. It contains information required for the execution of the currently active program; that is, it contains the current state of a CP. It has 16 bytes (128 bits). The PSW includes the instruction address, condition code, and other information used to control instruction sequencing and to determine the state of the CP. The active or controlling PSW is called the current PSW. It governs the program currently being executed.

Problem or supervisor bit mode (bit 15)
CP instructions can be classified as privileged and non-privileged. Note that, if misused, privileged instructions may damage system integrity and security. Privileged instructions should be executed only by z/OS programs, which are authorized. When the CP is in the supervisor state (bit 15 Off), it can execute any valid instruction. When the CP is in the problem state (bit 15 On), it can only execute non-privileged instructions. z/OS manages that, when its code is executing, bit 15 is Off; when an application program is executing, bit 15 is On. Some instructions are said to be semi-privileged, because they depend on other factors.

PSW key (bits 8-11)
The PSW key is used by a hardware mechanism within the CP called storage protection. It guarantees that programs running processes do not alter or access areas in storage that belong to other processes; refer to “Storage protection” on page 48 for more information.
1.13 Next sequential instruction address

Instruction address (bits 64 to 127) and CP interrupts

Bits 64 to 127 point to the storage address of the next instruction to be executed by this CP. When an instruction is fetched from central storage, its length is automatically added to this field. Then, PSW will point to the next instruction address. However, there are instructions as a BRANCH that may replace the contents of this field, pointing to the branched to instruction. The address contained in this PSW field may have 24, 31 or 64 bits, depending on the addressing mode attribute of the executing program. For compatibility reasons, old programs that still address small addresses are still allowed to execute. When in 24- or 31-bit addressing mode, the left-most bits of this field are filled with zeroes. Due to z/OS restrictions (field sizes on some control blocks), programs are required to run in 31 or 24 bit modes. In 64 bit mode, a program can address data (operands) above the bar, but the instructions of the program themselves must reside below the bar.

The CP has an interrupt capability, which permits it to switch rapidly to another program in response to exceptional conditions and external stimuli. When an interrupt occurs, the CP places the current PSW in an assigned storage location, called the old-PSW location, for the particular type of interrupt. The CP fetches a new PSW from a second assigned storage location. This new PSW determines the next program to be executed (FLIH). When it has finished processing the interrupt, the program handling the interrupt may reload the old PSW, making it the current PSW again, so that the interrupted program can continue. There are six types of interrupt: restart, external, supervisor call, program check, machine check, and I/O. Each type has a distinct pair of old-PSW and new-PSW locations permanently assigned in real storage.
1.14 Program-status-word format

PER mask - R (bit 1)
Bit 1 controls whether the CP is enabled for interrupts associated with program-event recording (PER). When the bit is zero, no PER event can cause an interruption. When the bit is one, interruptions are permitted, subject to the PER-event-mask bits in control register 9.

DAT mode - T (bit 5)
Bit 5 controls whether implicit dynamic address translation of logical and instruction addresses used to access storage takes place. When the bit is zero, DAT is off, and logical and instruction addresses are treated as real addresses. When the bit is one, DAT is on, and the dynamic-address-translation mechanism is invoked.

I/O mask - IO (bit 6)
Bit 6 controls whether the CP is enabled for I/O interruptions. When the bit is zero, an I/O interruption cannot occur in this CP. When the bit is one, I/O interruptions are subject to the I/O-interruption subclass-mask bits in control register 6. When an I/O-interruption subclass-mask bit is zero, an I/O interruption for that I/O-interruption subclass cannot occur; when the I/O-interruption subclass-mask bit is one, an I/O interruption for that I/O-interruption subclass can occur.

External mask - EX (bit 7)
Bit 7 controls whether the CP is enabled for interruption by conditions included in the external type. When the bit is zero, an external interruption cannot occur. When the bit is one, an
external interruption is subject to the corresponding external subclass-mask bits in control register 0; when the subclass-mask bit is zero, conditions associated with the subclass cannot cause an interruption; when the subclass-mask bit is one, an interruption in that subclass can occur.

**PSW key (bits 8-11)**
Bits 8-11 form the access key for storage references by the CP. If the reference is subject to key-controlled protection, the PSW key is matched with a storage key when information is stored or when information is fetched from a location that is protected against fetching. However, for one of the operands of each of MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH KEY, MOVE WITH SOURCE KEY, and MOVE WITH DESTINATION KEY, an access key specified as an operand is used instead of the PSW key.

**Architecture Mode bit (bit 12)**
This bit must be off (0) to indicate zArchitecture mode. When it is on (1), such as during IPL, the machine is running in ESA/390 mode.

**Machine-check mask - M (bit 13)**
Bit 13 controls whether the CP is enabled for interruption by machine-check conditions. When the bit is zero, a machine-check interruption cannot occur. When the bit is one, machine-check interruptions due to system damage and instruction-processing damage are permitted, but interruptions due to other machine-check-subclass conditions are subject to the subclass-mask bits in control register 14.

**Wait state - W (bit 14)**
When bit 14 is one, the CP is waiting; that is, no instructions are processed by the CP, but interruptions may take place. When bit 14 is zero, instruction fetching and execution occur in the normal manner. The wait indicator is on when the bit is one. When in wait state, the only way of getting out of such state is through an Interruption, which is covered in 1.20, “Interrupts” on page 39, or by an IPL (a z/OS “boot”).

Certain bits (6, 7 and 13), when off in the current PSW, place the CP in a disabled state; the CP does not accept Interrupts. So when z/OS, for any error reason (software or hardware) decides to stop a CP, it may set the PSW to the Disable and Wait state, forcing an IPL in order to restore the CP back to the running state.

**Problem state - P (bit 15)**
When bit 15 is one, the CP is in the problem state. When bit 15 is zero, the CP is in the supervisor state. In the supervisor state, all instructions can be executed, either non-privileged or privileged.

In the problem state, only those instructions that provide meaningful information to the problem program and that cannot affect system integrity are valid; such instructions are called unprivileged instructions; see “Problem or supervisor bit mode (bit 15)” on page 23.

The instructions that are never valid in the problem state are called privileged instructions. When a CP in the problem state attempts to execute a privileged instruction, a privileged-operation exception is recognized. Another group of instructions, called semi-privileged instructions, are executed by a CP in the problem state only if specific authority tests are met; otherwise, a privileged-operation exception or a special-operation exception is recognized; see “Problem or supervisor bit mode (bit 15)” on page 23.
Address-space control - AS (bits 16-17)
Bits 16 and 17, in conjunction with PSW bit 5 on (1), control the translation mode.
00 - Primary-space mode
01 - Access-register mode
10 - Secondary-space mode
11 - Home-space mode

Condition code - CC (bits 18-19)
Bits 18 and 19 are the two bits of the condition code. The condition code is set to 0, 1, 2, or 3, depending on the result obtained in executing certain instructions. Most arithmetic and logical operations, as well as some other operations, set the condition code. The instruction BRANCH ON CONDITION can specify any selection of the condition-code values as a criterion for branching.

The part of the CP that obtains results for the instructions is called the arithmetic and logic unit (ALU). The ALU has internally four bits that are set by certain instructions. At the end of these instructions, this 4-bit configuration is mapped into bits 18 and 19 of the current PSW.

As an example, the instruction COMPARE establishes a comparison between two operands. The result of the comparison is placed in the CC of the current PSW, as follows:
- If CC=00, then the operands are equal.
- If CC=01, then first operand is lower.
- If CC=10, then first operand is greater.
- CC=11 cannot occur after compare instructions.

To test the contents of a CC (set by a previous instruction), use the BRANCH ON CONDITION (BC) instruction. It contains an address of another instruction (branch address) to be executed, depending on the comparison of the CC and a mask M. The instruction address in the current PSW is replaced by the branch address if the condition code has one of the values specified by M; otherwise, normal instruction sequencing proceeds with the normal updated instruction address. Here are the types of codes:
- Condition code (bits 18 and 19 of the PSW) set by some instructions
- Return code - a code passed in the GPR 15 and associated with how a program ended
- Completion code - a code associated with how a task ended
- Reason code - a code passed in the GPR 0 detailing how a task ended

Program Mask (bits 20-23)
During the execution of an arithmetic instruction, the CP may find some unusual (or error) condition, such as overflows, lost of significance, or underflow. In such cases, the CP generates a program interrupt; refer to 1.22, “Types of interrupts” on page 43 for more details.

When this interrupt is treated by z/OS, usually the current task is abnormally ended (ABEND). However, in certain situations, programmers do not want an ABEND, so by using the unprivileged instruction SET PROGRAM MASK (SPM), they can mask such interrupts by setting some of the program mask bits to OFF. Each bit is associated with one type of condition:
- Fixed point overflow (bit 20)
- Decimal overflow (bit 21)
- Exponent underflow (bit 22)
- Significance (bit 23)
The active program is informed about these events through the condition code posted by the instruction where the events described happened.

The contents of the CP can be totally changed by two events:

- Loading a new PSW from storage along an interruption
- By executing the instruction LPSW, which copies 128 bits from memory to the current PSW

**Extended addressing mode - EA, BA (bits 31-32)**

The combination of bits 31 and 32 identify the addressing mode (24, 31 or 64) of the running program. Bit 31 controls the size of effective addresses and effective-address generation in conjunction with bit 32, the basic-addressing-mode bit. When bit 31 is zero, the addressing mode is controlled by bit 32. When bits 31 and 32 are both one, 64-bit addressing is specified; refer to 1.8, “Addressing mode” on page 16 for more information.
1.15 Prefixed save area (PSA)

Prefixed save area (PSA)

Figure 1-15 depicts the layout of the PSA in z/Architecture. The PSA maps the storage that starts at real storage location 0, for the related CPU. The function of the PSA is to map fixed hardware and software storage locations for the related CPU, each one having its own PSA.

IPL of a server

Initial program loading (IPL) provides a manual means for causing a program to be read from a designated device, and for initiating the execution of that program. At HMC one can choose which CPU will perform the IPL, usually it is CPU 0 by default.

An IPL is initiated manually by setting the load-unit-address controls to a four-digit number to designate an input device, and by subsequently activating the load-clear or load-normal function at the HMC. Activating the load-clear or load-normal sets the architectural mode to the ESA/390 mode, into which the Nucleus Initialization Program runs, until it issues a SIGP instruction to change the mode to zArchitecture.

The first CCW to be executed is an implied format-0 CCW, beginning in absolute location 0.
1.16 Several instruction formats

Instruction formats
An instruction is one, two, or three halfwords in length, as shown in Figure 1-16, and must be located in storage on a halfword boundary. Each instruction is in one of 26 basic formats: E, I, RI, RIE, RIL, RIS, RR, RRD, RRE, RRF, RRS, RS, RSI, RSL, RSY, RX, RXE, RXF, RXY, S, SI, SIL, SIY, SS, SSE, and SSF, with two variations of RRF, RS, RSY, RX, and RXY, three of RI and RIL, and six of RIE and SS.

Instruction set
The instruction set is a key item of the architecture, and it provides the function to allow programmers to access the hardware functions when creating programs. This means the set of instructions that programs (application or operating system) may use. In other words, the CP executes instructions, but only the ones defined by the computer architecture in the instruction set.

The quality of an architecture depends very much on how powerful the instruction set is in solving the various types of programming solutions. z/Architecture is a powerful architecture used to run in general purpose servers and to address different kinds of problems.

There are more than 900 instructions, as defined in z/Architecture. Each instruction has an implicit logic described in z/Architecture Principles of Operations, SA22-7832.
Instruction codes
Each instruction has the following generic information:

Op codes  The Operation codes, or instruction codes, indicate to the hardware which instruction to execute. For example, ADD (A) has an Op code of 5A; CHECKSUM (CKSM) has an Op code of B241.

Operands  The operands are the remainder of the instruction, following the Op code. They can be in storage (the instruction indicates the address), or in a register (the instruction indicates the register number).

In z/Architecture, an instruction may be two bytes, four bytes or six bytes in size, depending on the amount of information it needs to pass to the CP. The size is declared in the first two bits of the instruction code. For example, ADD REGISTER (1A), which adds the contents of two GPRs, only has two bytes: one for the code, and the other indicates the pair of GPRs involved. All instructions that process operands in storage need to address that operand. For that, z/Architecture adds the following:

- Contents of a GPR indicated in the instruction as a base register, such as B1 (as shown in Figure 1-16 on page 30.
- Contents of a GPR indicated in the instruction as an index or base register, such as X2 (as shown in Figure 1-16 on page 30.
- A displacement is indicated in the instruction, such as D1 (as shown in Figure 1-16 on page 30. For the RSY, RXY and SYI type of instruction, this displacement is 20 bits (1 MB range); for the others, it has 12 bits.

E  Denotes an operation using implied operands and having an extended Op code field.
I  Denotes the number of SVCs being called for.
RR  Denotes a register-and-register operation.
RRE  Denotes a register-and-register operation having an extended Op code field.
RRF  Denotes a register-and-register operation having an extended Op code field and an additional R field, or M field, or both.
RRS  Denotes a register-and-register operation having to be completed!
RRD  Denotes a register-and-register operation having to be completed!
RX  Denotes a register-and-indexed-storage operation.
RXE  Denotes a register-and-indexed-storage operation having an extended Op code field.
RXF  Denotes a register-and-indexed-storage operation having an extended Op code field and an additional R field.
RXY  Denotes a register-and-indexed-storage operation having a 20 bit displacement
RS  Denotes a register-and-storage operation.
RSE  Denotes a register-and-storage operation having an extended Op code field.
RSL  Denotes a storage operation (with an instruction format derived from the RSE format).
RSI  Denotes a register-and-immediate operation having an extended Op code field.
RSY  Denotes a register-and-storage operation having a 20 bit displacement
RI  Denotes a register-and-immediate operation having to be completed!
RIS  Denotes a register-and-immediate operation having to be completed!
RIE  Denotes a register-and-immediate operation having a longer extended Op code field.
RIL  Denotes a register-and-immediate operation having an extended Op code field and a longer immediate field.
S  Denotes a storage operand operation
SI  Denotes a storage-and-immediate operation.
SIL  Denotes a storage-and-immediate operation with a longer Operand.
SIY  Denotes a storage-and-immediate operation with 20 bit displacement.
SS  Denotes a storage-and-storage operation.
SSE  Denotes a storage-and-storage operation with an extended Operand.
SSF  Denotes a storage-and-storage operation having an extended Op code field and an additional R field, or M field, or both.
1.17 Microcode concepts

To better explain how some z/Architecture instructions are implemented in zSeries, we now introduce the concept of **microcode**. The vast majority of the z/Architecture instruction set is implemented through microcode. Microcode is a design option (not an architecture option) that is used to implement CP logic. To make it simple, we can divide the CP into two pieces: **data control** and **data flow** (also called ALU). Instructions are really executed in the data flow (where data is transformed); however, the sequence and timing of each of the multiple operations done in the data flow is ordered from the data control. It is similar to an orchestra: musicians (like pieces of data flow) know how to play their instrument, but they need guidance and tempo from the maestro (the data control) in order to execute properly.

In a microcoded CP, for each possible instruction of the instruction set, there is one micro program that tells data control what to do in order to execute the instruction in the data flow. The micro program has, in a special language, the sequence of orders to be sent by the data flow. These micro programs are loaded in a special internal memory in the CP, called control storage, at power-on reset time, by the Support Element. Decoding an instruction consists of finding the address in the control storage of its micro program. The opposite of microcoding is **hardwiring**, in which the logic of data control for each instruction is determined by Boolean hardware components. The advantage of microcoding is flexibility, where any correction or new function can be implemented by just changing or adding to the existent microcode. It is also possible that the same CP may switch instantaneously from one architecture to another (such as from ESA/390 to z/Architecture) by using another set of microcode to be loaded into a piece of its control storage.
1.18 z/Architecture components

Multiprocessing concepts
- Data formats
- Instruction set
- Interrupts
- Storage protection
- Addressing memory locations (virtual storage)
- CP signaling facility
- Time measurements and synchronization
- I/O operations
- Coupling Facility concepts

Figure 1-18 z/Architecture components

Multiprocessing concepts
Allowing many processes at the same time in a system can cause a single CP to be heavily utilized. In the 1970s, IBM introduced a tightly coupled multiprocessing complex, allowing more than one CP to execute more than one process (task) simultaneously. All these CPs share the same central storage (main storage, central storage, and real storage are different terms for the same kind of memory), which is controlled by a single z/OS copy in such storage.

To implement tightly coupled systems, the following items are needed in the architecture:
- Shared main storage, which allows several CPs (up to 80 with the z196, in some server models) to share the same main storage.
- CP-to-CP interconnection and signalling; refer to “CPU signaling facility” on page 66.
- Time-of-Day Clock (TOD) synchronization, to guarantee that all TODs in the same server are synchronized; for more details, refer to “Time measurement TOD” on page 67.
- Prefixing, implemented by the Prefix Register and related logic.

A system is made up of hardware products, including a central processor (CP), and software products, with the primary software being an operating system such as z/OS. Other types of software (system application programs, user application programmatic tools) also run on the system. The CP is the functional hardware unit that interprets and processes program instructions. The CP and other system hardware, such as channels and storage, make up a server complex.
The MVS operating system (z/OS) manages the instructions to be processed and the resources required to process them. When a single copy of the MVS operating system (MVS image) manages the processing of a CPC that has a single CP, the system configuration is called a **uniprocessor**.

When you add more CPs to the server complex, you add the capability of processing program instructions simultaneously. When all the CPs share central storage and a single MVS image manages the processing, work is assigned to a CP that is available to do the work. If a CP fails, work can be routed to another CP. This hardware and software organization is called a **tightly coupled multiprocessor**.

A tightly coupled multiprocessor has more than one CP, and a single MVS image, sharing central storage. The CPs are managed by the single MVS image, which assigns work to them. As mentioned, the multiprocessing facility includes the following:

- Shared main storage
- CP-to-CP interconnection
- TOD clock synchronization
- Prefixing

**Data formats**
The following data formats are used by z/Architecture:

**Decimal**
Decimal numbers may be represented in either the zoned or packed format. Both decimal number formats are of variable length; the instructions used to operate on decimal data each specify the length of their operands and results. Each byte of either format consists of a pair of four-bit codes; the four-bit codes include decimal-digit codes, sign codes, and a zone code.

**Floating point**
Four additional floating-point facilities improve the hexadecimal floating point (HFP) capability of the server and add a binary floating point (BFP from IEEE) capability and, for machines after the z9, add a decimal floating point (DFP) capability.

**Binary**
Binary integers are either **signed** or **unsigned**. Unsigned binary integers have the same format as signed binary integers, except that the left-most bit is interpreted as another numeric bit, rather than a sign bit. There is no complement notation because all unsigned binary integers are considered positive.

**Alphanumeric**
Such as EBCDIC, ASCII, UNICODE, UTF-8.

**Note:** For an illustration of the data formats, refer to Figure 1-19 on page 36.

**Instruction set**
The instruction set for z/Architecture has many new instructions, and many of them operate on 64-bit binary integers. All ESA/390 instructions, except for the asynchronous-pageout, asynchronous-data-mover, program-call-fast, and vector facilities, are included in z/Architecture. The bit positions of the general registers and control registers of z/Architecture are numbered 0-63. An ESA/390 instruction that operates on bit positions 0-31 of a 32-bit register in ESA/390, operates instead on bit positions 32-63 of a 64-bit register in z/Architecture. The other bits of the register, 0-31, are left intact.

**Interrupts**
With an interrupt, certain events may cause a change in the status of the server and the selection of the next instruction to be executed by the server.
Storage protection
Storage protection mechanisms are needed because several programs from different users (processes), may occupy locations in the same memory, at the same time. This facility is complemented by the Virtual Mode (PSW bit 5 on), since the majority of the tasks (hence, users), will run with the same Storage Protection Key: 8.

Addressing memory locations
z/Architecture contains a scheme to allow programs to address memory locations, where data and instructions are stored (virtual storage).

CP signalling facility
The signaling facility among CPs consists of a CP-signaling-and-response facility that uses the SIGNAL PROCESSOR order and a mechanism to interpret and act on several order codes. The facility provides for communications among CPs, including transmitting, receiving, and decoding a set of assigned order codes, initiating the specified operation, and responding to the signaling CP. A CP can address a SIGNAL PROCESSOR instruction to itself.

Time measurements and synchronization
Time measurements and time synchronization are provided, as every CP has access to a TOD clock and every channel subsystem has access to at least one channel-subsystem timer. When multiple channel-subsystem timers are provided, synchronization among these timers is also provided, creating the effect that all the timing facilities of the channel subsystem share a single timer. Synchronization among these timers may be supplied either through a TOD clock, or independently, by the channel subsystem.

Coupling Facility concepts
The Coupling Facility enables high performance data sharing among z/OS systems that are connected by means of the facility. The Coupling Facility provides storage that can be dynamically partitioned for caching data in shared buffers, maintaining work queues and status information in shared lists, and locking data by means of shared lock controls. z/OS services provide access to and manipulation of Coupling Facility contents.
1.19  z/Architecture data formats

Data formats
Central storage is a magnetic digital device used to store instructions and data manipulated by such instructions. Central storage is made up of bytes, and each byte in z/Architecture is eight bits. The format of the data allowed depends on how the logic of an instruction understands it. The following data formats are used with z/Architecture.

Binary integers
Binary integers (also called fixed point) are treated as signed or unsigned, without a fraction point. In an unsigned binary integer, all bits are used to express the absolute value of the number. When two unsigned binary integers of different lengths are added, the shorter number is considered to be extended on the left with zeros. For signed binary integers, the left-most bit represents the sign (0 for positive and 1 for negative), which is followed by the numeric field. Positive numbers are represented in true binary notation with the sign bit set to zero. Negative numbers are represented in two's-complement binary notation, with a one in the sign-bit position. The length of such data can be two bytes (a half word), four bytes (a full word) or eight bytes (a double word). Some Floating Point instructions use extended operands made of sixteen bytes (a quad word).

Decimal numbers
Decimal numbers are represented in a variable number of bytes, and may be represented in either the zoned or packed format. Each byte of either format consists of a pair of four-bit codes. The four-bit codes may be decimal digit codes, sign codes, and a zone code.
In the zoned format, the rightmost four bits of a byte are called the *numeric bits* (N), and normally consist of a code representing a decimal digit (from 0 to 9). The leftmost four bits of a byte are called the *zone bits* (Z), usually an X’F’, except for the rightmost byte of a decimal operand, where these four bits are treated as a sign (S).

Decimal digits in the zoned format may be part of a larger character set (as EBCDIC), which includes also alphabetic and special characters. The zoned format is, therefore, suitable for input, editing, and output of numeric data in human-readable form. There are no decimal-arithmetic instructions that operate directly on decimal numbers in the zoned format; such numbers must first be converted to the packed decimal format.

In the packed format, each byte contains two decimal digits (D), except for the rightmost byte, which contains a sign to the right of a decimal digit (Hex C or Hex F for positive, and Hex D or Hex B for negative (Hex A and Hex E are also positive, but seldom used)). Decimal arithmetic operation is performed with operands in the packed format, and generates results in the packed format. The packed-format operands and results of decimal-arithmetic instructions may be up to 16 bytes (31 digits and sign). The editing instructions can fetch as many as 256 bytes from one or more decimal numbers of variable length, each in packed format. There are instructions to convert between the numeric data formats.

**Floating point numbers**

The floating-point number is used to represent large real numbers with high precision, which is usually needed in engineering and science processing. This format includes a fraction point separating the integer part from the rest. It has three components: a sign bit, a signed binary exponent, and a significant. The significant consists of an implicit unit digit to the left of an implied radix point, and an explicit fraction field to the right. The significant digits are based on the radix, 2 or 16. Decimal Floating Point numbers need a table to interpret them.

The magnitude (an unsigned value) of the number is the product of the significant and the radix raised to the power of the exponent. The number is positive or negative depending on whether the sign bit is zero or one, respectively. The radix values 16 and 2 lead to the terminology “hexadecimal” and “binary” floating point (HFP developed by IBM and BFP developed by IEEE). The formats are also based on three operand lengths: short (32 bits), long (64 bits), and extended (128 bits). There are instructions able to execute both types, just as there are instructions specialized in just one of the formats.

The exponent of an HFP number is represented in the number as an unsigned seven-bit binary integer called the *characteristic*. The characteristic is obtained by adding 64 to the exponent value (excess-64 notation). The range of the characteristic is 0 to 127 (7 bits), which corresponds to an exponent range of -64 (1111111) to +63 (0111111).

The exponent of a BFP number is represented in the number as an unsigned binary integer called the biased exponent. The biased exponent is obtained by adding a bias to the exponent value. The number of bit positions containing the biased exponent, the value of the bias, and the exponent range depend on the number format (short, long, or extended) and are shown for the three formats. For more information about floating point representation, refer to “Floating point registers” on page 22.

**EBCDIC data**

Extended Binary Coded Decimal Interchange Code (EBCDIC) is an alphanumeric 8 bit (256 possibilities) code, developed by IBM, to represent in memory graphic signs as letters, numbers, and certain special signals such as: $, #, & and so on. For instance, if you key the letter A in your mainframe keyboard, the byte in memory where that letter is read presents the hexadecimal pattern C1 (1100 0001 in binary) when in EBCDIC.
If you key the letter B, you have C2 (1100 0010). If you key the number 2, you have F2 (1111 0010). Refer to *z/Architecture Reference Summary, SA22-7871* for the complete set of EBCDIC code.

**ASCII data**
American Standard Code for Information Interchange (ASCII) is also an alphanumeric 8-bit code. It is fully supported in *z/Architecture* by a set of several instructions.

**Unicode**
Unicode an alphanumeric double byte code with 64 K possibilities. It is fully supported in *z/Architecture* by a set of several instructions.

**UTF-8**
UTF-8 is an alphanumeric quad byte code with 4 G possibilities. It is fully supported in *z/Architecture* by a set of several instructions.
1.20 Interrupts

Interrupt events
An interrupt occurs when the CP detects one of six events. During interrupt processing, the CP does the following (this is called PSW Swap in some IBM Manuals):

- Stores (saves) the current PSW in a specific central storage location named old PSW.
- Fetches, from a specific central storage location named new PSW, an image of the PSW and loads it in the current PSW.
- Stores information identifying the cause of the interrupt in specific central storage locations, called interrupt codes.

Note: The old and new PSWs are just copies of the current PSW contents. Processing resumes as specified by the new PSW instruction address and status. The old PSW stored on an interrupt normally contains the status and the address of the instruction that would have been executed next had the interrupt not occurred, thus permitting later the resumption of the interrupted program (and task).

Interrupt event groups
Six groups of events cause interrupts, as follows:

- Restart (also called PSW restart at the HMC)
- External
For each type of interrupt there is, in central storage, a trio of locations for old PSW, new PSW, and interrupt codes. These locations are kept in an 8-KB area at the very beginning of the real storage for each processor, called the Prefix Storage Area (PSA). For example, the address of the SVC New PSW is X'1C0', the address of SVC Old PSW is X'140' and the address of the SVC interrupt code is X'88'. For more information, refer to “Fixed Storage Locations” in SA22-7871.
1.21 Interrupt processing

Interrupts
Usually, a CP is temporarily disabled by z/OS because of integrity reasons, as described by bits 6, 7 and 13 in the PSW and also by bit settings in the CRs. In this case, other interrupt are not lost, but stacked in the original hardware element, or handled by other CPs in the server.

Types of interrupts
As mentioned, there are six types of interruption conditions: restart, external, supervisor call (SVC), program check, machine check, and I/O.

Reasons for interrupts
When the CP finishes the execution of one instruction, it executes the next sequential instruction (the one located at an address after the one just executed). The instruction address field in the current PSW is updated in order to execute the next instruction. If the logic (set of logically connected instructions) of the program allows it, the next instruction can branch to another instruction through the BRANCH ON CONDITION instruction.

In a sense, an interrupt is a sort of branching, but there is a logical difference between a BRANCH instruction issued in a program and an interrupt. A BRANCH is simply a twist in the logic of the program. In an interrupt, however, one of the six interruption conditions occurred which needs to be brought to the attention of z/OS immediately. In the following section, we describe the flow of an interrupt.
Step 1
One of the six types of interrupt is brought to the attention of the hardware. Therefore, the running program must be interrupted by means of a PSW Swap. If the actual state of the running program in the PSW is disabled, this interrupt is stacked in the hardware itself and will be taken care of as soon as this, or another CPU, enters an enabled state.

Step 2 and Step 3
The CP was following the sequence of the instructions pointed to by the instruction address in the current PSW and suddenly, after the interrupt, it now addresses (and executes) the instruction pointed to by the copy of PSW located in the new PSW, which is now loaded in the current PSW. All the new PSWs point to one of the six first level interrupt handlers (FLIH) whose duty is to save the running program status, and then select an appropriate Service Routine, also called second level interrupt handler (SLIH) to pass control of the CPU.

Each type of interrupt has two related PSWs, called old and new, in permanently assigned real storage locations. Each type of interrupt involves storing information that identifies the cause of the interrupt, storing the current PSW at the old PSW location, and fetching the PSW at the new PSW location, which becomes the current PSW.

**Note:** All the events generating interrupts have something in common: they cannot be processed by an application program. Instead, they need z/OS services (see Step 4). So why is it that a simple branch to z/OS code does not solve the problem? The reason is because a branch cannot, and does not, change the status bits of the current PSW.

Step 4
Control is then passed to the first level interrupt handler (FLIH). z/OS will use privileged instructions to respond to the event; the new PSW (prepared by z/OS itself during NIP processing) has bit 15 turned off (see "Problem state - P (bit 15)" on page 26). z/OS needs to access some storage locations to respond to the event and save the previous status, and the new PSW (prepared by z/OS) has the PSW key set for those storage locations.

Step 5
After saving the status before the interrupt, control is passed, for each type of interrupt, to a second level interrupt handler for further processing of the interrupt. After servicing the interrupt, the SLIH calls the z/OS dispatcher.

Step 6
The z/OS dispatcher that dispatches all waiting tasks can dispatch the interrupted program if an available CP is ready for new work.

Step 7
The z/OS dispatcher does this by using the old PSW, which has been saved, and which contains CP status information necessary for resumption of the interrupted program. Then the instruction LOAD PSW EXTENDED may be used to restore the current PSW to the value of the old PSW, and return control to the interrupted program (this usually happens in the case of SRBs). If FLIH had to save the status on Control Blocks, as it might in the case of TCBs and saved the old PSW in the program request block (PRB), the dispatcher will then use a LPSW instruction instead, because the Control Blocks have only an ESA/390 image of the PSW. Please see the differences among these 2 instructions at the POP on pages 10-47 and 10-48.

In the following section, we examine each type of interrupt in more detail.
1.22 Types of interrupts

- Program check
- Supervisor call
- I/O
- External
- Machine check
- Restart

Program check interrupt
This interrupt is generated by the CP when something is wrong during the execution of an instruction. For example:

- 0001 is an operation exception meaning that the CP tried to execute an instruction with an unknown operation code.
- 0002 is a privileged operation exception meaning that the CP tried to execute a privileged instruction with the current PSW having bit 15 on, indicating problem mode.

Usually the z/OS reaction to a program check interrupt is to ABEND (abnormally terminate) the task that was executing the program which issued the instruction in error.

Remember that certain causes for program interrupts can be masked by the SET PROGRAM MASK (SPM) instruction, as covered in “Program-status-word format” on page 25. However, getting something wrong during instruction execution does not necessarily indicate an error. For example, a page fault program interrupt (interrupt code x’11’ indicates that the virtual storage address does not correspond to a real address in central storage, but the task is not ABENDed. Refer to “Dynamic address translation (DAT)” on page 57 for more information.

Sometimes a bad pointer used in a branch instruction may cause operation or protection exception program interrupts. These are difficult to diagnose since there is no clue about how the system got there. With the new wild branch hardware facility of the z9-109, the last address from which a successful branch instruction was executed is kept in storage location x’1110’. 

Figure 1-22 Types of interrupts
1.23 Supervisor call interrupt

Supervisor call interruption

This interrupt is triggered in the CP by the execution of the SUPERVISOR CALL (SVC) instruction. This instruction has two bytes:

- 0A as the Op code in the first byte
- The SVC interrupt code (SVC number) in the second byte

When executed by the CP, the SVC instruction causes an SVC interrupt; that is, the current PSW is stored in the PSA at the SVC old PSW, a new PSW from the PSA is loaded in the current PSW, and the second byte of the instruction is stored in the SVC interrupt code in PSA memory.

The reason for such interrupts is part of the architecture, where an application program running in problem mode (bit 15 of the current PSW is on) may pass control to z/OS asking for a service. After the interrupt, the CP goes to the current PSW to get the address of the next instruction. Now, the content of the current PSW is the copy of the SVC new PSW, which is in supervisor state. This PSW points to the first instruction of a code inside z/OS named SVC first level interrupt handler (FLIH). This FLIH, already running in supervisor state, saves the contents of the registers in the Task Control Block (TCB), a control block in memory allocated by z/OS to keep the state of the interrupted process, and saves an ESA/390 image of the SVC old PSW in the Program Request Block (PRB), pointed to by the first word of the TCB. In the majority of the cases, SVC FLIH branches to a second level interrupt handler (SLIH) z/OS routine. SLIH gets the interruption code (SVC number), that is used as an index into an SVC
table control block.

There is an entry in such a table for every possible SVC number. The entry describes the state and the address of a z/OS SVC routine that handles a required function associated with the SVC number. For example, SVC 0 means the z/OS EXCP routine, the one in charge of starting an I/O operation. Consulting the proper entry in the SVC table, the SVC SLIH routine branches to the specific SVC routine. Consequently, the program issuing the SVC instruction needs to know the relationship between the SVC interrupt code and the z/OS component to be invoked. For example:

- 00 means that the application program is invoking the input/output supervisor (IOS), asking for an I/O operation (EXCP).
- 01 means that the active task wants to enter the wait state (Wait).
- 04 means that the active task wants the permit to address some virtual addresses (GETMAIN). This means to allocate some amount of virtual storage, which can also be done with SVCs 10 and 120.

However, the limit for the number of SVC routines is 256, because the SVC number has just one byte. To circumvent the problem, z/OS introduced the concept of Extended SVC Router (ESR). It implies that certain SVC routines, such as SVC 109, are just another router. By inspecting the contents of the general purpose register (GPR) 1 and using another table, SVC routine 109 routes to a z/OS routine that will execute the required function. For example, depending on the contents of GPR 1, SVC 109 may invoke IFAUSAGE, or MSGDISP, or OUTADD z/OS routines.

After the request is processed by the z/OS SVC routine, the interrupted program can regain control by restoring of its registers and the LPSW instruction issued against the ESA/390 copy of the SVC old PSW in the PRB, pointed to by the first fullword of the TCB.

**Input/output interrupt**

An I/O operation is requested by a task to the input/output supervisor (IOS) through an SVC 0 instruction. After the SVC interrupt processing, the SVC FLIH passes control to IOS. In z/Architecture, the I/O operation is not handled by the CP executing z/OS code. There are less expensive and more specialized servers to do the job: the channels. When IOS issues the privileged START SUBCHANNEL (SSCH) instruction, the CP delegates it to a SAP that will find a channel for the execution of the I/O operation. Then, the I/O operation is a dialogue between the channel and an I/O control unit, in order to move data between central storage and the I/O device controlled by such a controller. After the execution of the SSCH instruction, IOS returns control to the task issuer of the SVC 0. If needed, the access method of this task places itself in wait, till the end of the I/O operation.

Now, how does IOS and the CP become aware that the I/O operation handled by the channel is finished? This is handled through an I/O interrupt triggered by a SAP, after receiving the device end, control unit end and channel end from the channel. The I/O new PSW points to the IOS code in z/OS (I/O FLIH) and the interrupt codes tell IOS which device has an I/O operation that has completed. Also, be aware that there may be many I/O operations running in parallel. The final status of the I/O operation is kept in a control block called the Interrupt Request Block (IRB), which is placed in storage through the execution of the TEST SUBCHANNEL instruction issued by IOS. The I/O old PSW has the current PSW at the moment of the I/O interrupt, so it can be used to resume the processing of the interrupted task. This return to the interrupted code is done, of course, by the dispatcher, gathering the status saved in some control blocks, and restoring it in the proper hardware registers.
**External interrupt**

This type of interrupt has eight different causes, usually not connected with what the active program is doing, as follows:

- **0040 Interrupt key** - An interrupt request for the interrupt key is generated when the operator activates that key in the HMC console; refer to “System components” on page 7.
- **1004 Clock comparator** - The contents of the TOD Clock became equal to the Clock Comparator; refer to “Time measurement TOD” on page 67.
- **1005 CPU timer** - The contents of the CPU Timer became negative; refer to “Time measurement (CP timer)” on page 69.
- **1200 Malfunction alert** - Another CPU in the MP tightly coupled complex is in check stop state due to an hardware error. The address of the CPU that generated the condition is stored at PSA locations 132-133.
- **1201 Emergency signal** - Generated by the SIGNAL PROCESSOR instruction when z/OS, running in a CPU with a hardware malfunction, decided to stop (Wait Disable) that CPU. The address of the CPU sending the signal is provided with the interrupt code when the interrupt occurs. (Note that the CPU receiving such an interrupt is not the one with the defect.)
- **1202 External call** - Generated by the SIGNAL PROCESSOR instruction when a program wants to communicate synchronously or asynchronously with another program running in another processor. The address of the processor sending the signal is provided with the interrupt code when the interrupt occurs.
- **1406 ETR** - An interrupt request for the External Timer Reference (ETR) is generated when a port availability change occurs at any port in the current server-port group, or when an ETR alert occurs; refer to “Time measurement TOD” on page 67.
- **2401 Service signal** - An interrupt request for a service signal is generated upon the completion of certain configuration control and maintenance functions, such as those initiated by means of the model-dependent DIAGNOSE instruction. A 32-bit parameter is provided with the interrupt to assist the program in determining the operation for which the interrupt is reported.

**Machine check interrupt**

This type of interrupt is a part of the machine check-handling mechanism. This mechanism provides extensive equipment malfunction detection to ensure the integrity of system operation and to permit automatic recovery from some malfunctions. This detection design is mainly based on the redundancy of components. For example, within some PUs, there are execution units, two of them executing the same instruction and a third comparing the results.

Equipment malfunctions and certain external disturbances are reported by means of a machine check interrupt to assist z/OS in program damage assessment and recovery. The interrupt supplies z/OS with information about the extent of the damage and the location and nature of the cause. Four hardware mechanisms may be used to provide recovery from server-detected malfunctions: error checking and correction, CP retry, channel subsystem recovery, and unit deletion.

There are two types of machine check interrupt conditions: *exigent* conditions and *repressible* conditions:

- Exigent machine check interrupt conditions are those in which damage has or would have occurred such that execution of the current instruction or interrupt sequence cannot safely continue.
- Repressible machine check interrupt conditions are those in which the results of the instruction processing sequence have not been affected.
Restart interrupt
The restart interrupt provides a means for the operator (by using the restart function at HMC) or a program running on another CP (through a SIGNAL PROCESSOR instruction) to invoke the execution of a specified z/OS component program. The CP cannot be disabled for this interrupt. In z/OS, the specific component does an evaluation of the system status, reporting hangs, locks, and unusual states in certain tasks. It gives the operator a chance to cancel the offending task. It maybe the last chance in order to avoid an IPL.
1.24 Storage protection

Storage protection
This is one of the mechanisms implemented by z/Architecture to protect central storage. With multiprocessing, hundreds of tasks can run programs accessing physically any piece of central storage.

Storage protection imposes limits and a task is only able to access (for read or write) the central storage locations with its own data and programs, or, if specifically allowed, to read areas from other tasks. Any violation of this rule causes the CP to generate a program interrupt 0004 (protection exception), that makes z/OS ABEND that task.

All real addresses manipulated by CPs or channels must go through the storage protection verification before being used as an argument to access the contents of central storage. The input of storage protection is a real storage address, and the output is an OK or a program interrupt 0004.

Storage key
For each 4 KB block of central storage, there is a 7-bit control field, called a storage key. This key is used as follows:

- Access control bits

  Bits 0-3 are matched against the 4-bit protection key in the current program status word (PSW) bits 8-11 whenever information is stored, or whenever information is fetched from a location that is protected against fetching.
PSW protection keys

There are 16 protection keys (0 to 15) provided by the PSW, and they are matched against the access control bits in the storage key. The storage protection implementation formats central (real) storage into 4-KB frames. For each 4-KB frame, there is a 7-bit storage key. Each storage key has:

- 4 access control bits (shown as KKKK in Figure 1-24 on page 48)
- Fetch bit
- Reference bit
- Change bit

PSW key field

The PSW key field (bits 8 to 11) is set by the privileged SET PSW KEY FROM ADDRESS instruction. This key is compared to the access control bits (kkkk) for the frame being referenced for storage access.

Fetch protection bit

Bit 4 indicates whether protection applies to fetch-type references. A zero indicates that only store-type references are monitored, and that fetching with any protection key is permitted; a one indicates that protection applies to both fetching and storing. No distinction is made between the fetching of instructions and the fetching of operands.

Reference bit

Bit 5 is associated with dynamic address translation (DAT). It is normally set to one whenever a location in the related 4 KB storage block is referred to for either storing or fetching of information.

Change bit

Bit 6 is also associated with DAT. It is set to one each time that information is stored into the corresponding 4 KB block of storage.

Note: For further information about storage protection, refer to “Storage protection logic” on page 50.
1.25 Storage protection logic

Storage protection logic

z/OS may alter storage key bits by issuing the SSKE instruction, and may inspect them by the ISKE and IVSK instructions. The reference bit is inspected and switched off after inspection by the RRBE instruction. On top of that, the CPU storage hardware switches on the reference bit when the frame is accessed by any CPU or any channel, and also switches on the change bit when the frame contents are changed by those components. The reference and change bits do not participate in the storage protection algorithm. They are used for virtual storage implementation; refer to “Dynamic address translation (DAT)” on page 57.

The following conclusions can be reached from the logic:

- If a running program has PSW key equal to 0000, it may access any frame in memory.
- If the fetch bit is off in a frame, any program can read the contents of that frame.
- To read the contents of a frame where the fetch bit is on, the PSW key of the running program must match the access control 4 bits in the storage key of the frame.
- To alter (write) the contents of a frame, the PSW key of the running program must match the access control 4 bit in the storage key of the frame.

z/OS exploits storage protection by managing frame storage key values and running the program PSW key field in the current PSW; for example, several z/OS routines run with PSW key zero, and others run with PSW key one. Application code has PSW key eight.
1.26 Addresses and address spaces

Evolution of architectures

z/Architecture is the next step in the evolution from the System/360 to the System/370, System/370 extended architecture (370-XA), Enterprise Systems Architecture/370 (ESA/370), and Enterprise Systems Architecture/390 (ESA/390). z/Architecture includes all of the facilities of ESA/390 except for the asynchronous-pageout, asynchronous-data-mover, program-call-fast, and vector facilities. z/Architecture also provides significant extensions, as follows:

- 64-bit general registers and control registers, a 64-bit addressing mode, in addition to the 24-bit and 31-bit addressing modes of ESA/390, which are carried forward to z/Architecture.

  Both operand addresses and instruction addresses can be 64-bit addresses. The program status word (PSW) is expanded to 16 bytes to contain the larger instruction address. The PSW also contains a newly assigned bit that specifies 64-bit addressing mode.

- Up to three additional levels of dynamic-address-translation (DAT) tables, called region tables, for translating 64-bit virtual addresses.

In order to support the growth in e-business with large numbers of users and transactions, a 64-bit virtual addressing scheme has been introduced. The first basic support in the z/OS operating system for 64-bit virtual addressing was introduced with z/OS V1R2.
An address is an argument used by the CP to access the contents of bytes in memory or cache. A CP needs to access memory (also called central or real storage) to reach:

- An instruction, where its address is in the current PSW; refer to “Current program-status word (PSW)” on page 23.
- A memory operand referred to by an RS, RX, SS type of instruction. The operand address is formed by:
  - Adding the contents of a GPR, named base register plus a displacement of 12 bits (or 20 bits) declared in the instruction (refer to “Several instruction formats” on page 30), or
  - Adding the contents of a GPR, named base register plus the contents of a GPR, named index plus a displacement of 12 bits declared in the instruction.

**Addresses**

An address size refers to the maximum number of significant bits that can represent an address. With z/Architecture, three sizes of addresses are provided:

- **24-bit** A 24-bit address can accommodate a maximum of 16,777,216 (16M) bytes.
- **31-bit** With a 31-bit address, 2,147,483,648 (2 G) bytes can be addressed.
- **64-bit** With a 64-bit address, 8,446,744,073,709,551,616 (16 E) bytes can be addressed.

Any program running with 24-bit or 31-bit addresses can run in z/Architecture.
1.27 z/Architecture address sizes

Addresses in 4-byte fields
The bits of an address that is 31 bits regardless of the addressing mode are numbered 1-31. When a 24-bit or 31-bit address is contained in a four-byte field in storage, the bits are numbered 8-31 or 1-31, respectively.

A 24-bit or 31-bit virtual address is expanded to 64 bits by appending 40 or 33 zeros, respectively, on the left before it is translated by means of the DAT process. A 24-bit or 31-bit real address is similarly expanded to 64 bits before it is transformed by prefixing. A 24-bit or 31-bit absolute address is expanded to 64 bits before main storage is accessed. Thus, a 24-bit address always designates a location in the first 16 MB block of the 16 exabyte storage addressable by a 64-bit address, and a 31-bit address always designates a location in the first 2 GB block.

64-bit words
The bits of a 24-bit, 31-bit, or 64-bit address produced by address arithmetic under the control of the current addressing mode are numbered 40-63, 33-63, and 0-63, respectively, corresponding to the numbering of base address and index bits in a general register.

Therefore, whenever the server generates and provides to the program a 24-bit or 31-bit address, the address is made available (placed in storage or loaded into a general register) by being imbedded in a 32-bit field, with the left-most eight bits or one bit in the field, respectively, set to zeros. When the address is loaded into a general register, bits 0-31 of the register remain unchanged.
1.28 Storage addressing

Accessing storage
Storage is viewed as a long, horizontal string of bits, and is available in multiples of 4 KB blocks. For most operations, access to storage proceeds in a left-to-right sequence. The string of bits is subdivided into units of eight bits. This eight-bit unit is called a byte, which is the basic building block of all information formats.

Bits and bytes
Each byte location in storage is identified by a unique nonnegative integer, which is the address of that byte location or, simply, the byte address. Adjacent byte locations have consecutive addresses, starting with 0 on the left and proceeding in a left-to-right sequence. Addresses are unsigned binary integers and are 24, 31, or 64 bits.

The value given for a byte is the value obtained by considering the bits of the byte to represent a binary code. Thus, when a byte is said to contain a zero, the value 00000000 binary, or 00 hex, is meant.

Within each group of bytes, bits are numbered in a left-to-right sequence. The left-most bits are sometimes referred to as the “high-order” bits and the right-most bits as the “low-order” bits. Bit numbers are not storage addresses, however. Only bytes can be addressed. To operate on individual bits of a byte in storage, it is necessary to access the entire byte.

The bits in a byte are numbered 0 through 7, from left to right.
1.29 Real storage locations

Storage location
Central storage is also referred to as main storage or real storage. When a program is executing in an address space, the program’s virtual storage address in a 4 K frame where the execution is taking place must be located in a central storage frame, as shown in Figure 1-29.

Address space references
Main storage provides the system with directly addressable fast access storage of data. Both data and programs must be loaded into main storage (from input devices) before they can be processed. The storage is available in multiples of 4 KB blocks.

Therefore, MVS programs and data reside in virtual storage that, when necessary, is backed by central storage. Most programs and data do not depend on their real addresses. Some MVS programs, however, do depend on real addresses and some require these real addresses to be less than 16 megabytes. MVS reserves as much central storage below 16 megabytes as it can for such programs and, for the most part, handles their central storage dependencies without requiring them to make any changes.

Virtual address
A virtual address identifies a location in virtual storage. When a virtual address is used for an access to main storage, it is translated by means of dynamic address translation to a real address, which is then further converted by prefixing to an absolute address.
A real address identifies a location in real storage. When a real address is used for an access to main storage it is converted, by means of prefixing, to an absolute address. At any instant there is one real address-to-absolute address mapping for each CP in the configuration. When a real address is used by a CP to access main storage, it is converted to an absolute address by prefixing.

Storage consisting of byte locations sequenced according to their real addresses is referred to as **real storage**.

**Virtual address to main storage**

When a virtual address is used by a CP to access main storage it is first converted, by means of dynamic address translation (DAT), to a real address, and then, by means of prefixing, to an absolute address. DAT may use from five to two levels of tables (region first table (R1 index), region second table, region third table, segment table, and page table) as transformation parameters.

The designation (origin and length) of the highest-level table for a specific address space is called an address-space-control element, and it is found for use by DAT in a control register or as specified by an access register. Alternatively, the address-space-control element for an address space may be a real space designation, which indicates that DAT is to translate the virtual address simply by treating it as a real address and without using any tables.
1.30 Dynamic address translation (DAT)

**Dynamic address translation**
Dynamic address translation (DAT) is the process of translating a virtual address during a storage reference into the corresponding real address.

A segment table designation or region table designation causes translation to be performed by means of tables established by the operating system in real or absolute storage.

In the process of translation when using a segment table designation or a region table designation, three types of units of information are recognized:

- **Region**: A block of sequential virtual addresses spanning 2 GB and beginning at the 4-GB boundary (above the bar).
- **Segment**: A block of sequential virtual addresses spanning 1 MB and beginning at a 1 MB boundary.
- **Page**: A block of sequential virtual addresses spanning 4 KB and beginning at a 4-KB boundary. Starting with the z10 EC server, optionally a page can have a size of 1-MB addresses (large pages). However, this is only possible for pages above the bar. All those pages are fixed in central storage.

**z/OS translation tables**
There are three levels of translating tables: segment table, region table, and page table. Each entry in the segment table points to a page table. Each entry in the page table points to the location of the frame associated with that page.

---

*Figure 1-30  Dynamic address translation (DAT)*

Location = 8000 + A26 = 8A26 (the location 8000 was taken from the page table entry)
Dynamic address translation performs the following tasks:

- Receives an address from the CP (it does not matter whether it refers to an operand or to an instruction).
- Divides the address by 1 M. The quotient is the number of the segments (S), and the rest is the displacement within the segment (D1).
- Finds the corresponding entry in the segment table to obtain the pointer of the corresponding page table.
- Divides the D1 by 4 K. The quotient is the number of the page (P), and the rest is the displacement within the page (D2). It finds the corresponding entry for P2 in the page table, getting the location of the corresponding frame.
- Adds D2 with the frame location and passes back this result to the CP to allow access to the memory contents.

Figure 1-31 on page 59 illustrates the process of translating the address x ‘4A6C8A26’.
1.31 Dynamic address translation

Control registers and DAT
Translation of a virtual address is controlled by the DAT-mode bit and address-space-control bits in the PSW and by the address-space-control elements (ASCEs) in control registers 1, 7, and 13, and as specified by the access registers. When the ASCE used in a translation is a region-first-table designation, the translation is performed by means of a region first table, region second table, region third table, segment table, and page table, all of which reside in real or absolute storage.

When the ASCE is a lower-level type of table designation (region-second-table designation, region-third-table designation, or segment-table designation), then the translation is performed by means of only the table levels beginning with the designated level—and the virtual-address bits that would, if nonzero, require use of a higher level or levels of table must be all zeros; otherwise, an ASCE-type exception is recognized. When the ASCE is a real space designation, the virtual address is treated as a real address, and table entries in real or absolute storage are not used.

Page faults
Following the DAT attempt, either a real address is determined or a page fault occurs. There is a bit in the page table entry called the invalid bit. When the invalid bit is on, it means that the content of the referenced page is not mapped into a frame in real storage. DAT reacts to this by generating a program interrupt code X’11” indicating a page fault. This page must be read in from external storage called a page data set.
1.32 Page faults

Page data sets
Paging data sets contain the paged-out portions of all virtual storage address spaces. In addition, output to virtual I/O devices may be stored in the paging data sets. Before the first IPL, an installation must allocate sufficient space on paging data sets to back up the following virtual storage areas:

- Primary storage for the pageable portions of the common area
- Secondary storage for duplicate copies of the pageable common area
- The paged-out portions of all swapped-in address spaces—both system and installation
- Space for all address spaces that are, or were, swapped out
- VIO data sets that are backed by auxiliary storage

Address space virtual storage
Every address space has the same virtual storage mapping. Figure 1-32 shows two address spaces with a virtual storage address of 10254000 which contains either data or executable code belonging to that address space. For each address space to reference the 4 K page beginning at that address, that 4 K page must reside in central storage, as shown by the real addresses in central storage. When referenced, the page could either be in central storage already, or a page fault occurs and it must be read from the page data set where it resides.

DAT and MVS
To determine how to access the page in virtual storage, the functions are divided between hardware (DAT) and the operating system (z/OS), as follows:
DAT is in charge of translating the address, from now on called the **virtual address**, producing a location, from now on called the **real address in central storage**, or generating a page fault. 

There is a bit in the current PSW (bit 5) that when on, DAT is implicitly invoked for each address referenced by the CP. When off, it means that the virtual address is equal to the real address and DAT is not invoked.

**MVS is in charge of the following:**
- Maintaining the tables (Segment and Page tables).
- Deciding which page contents are kept in real storage frames.
- Allocating I/O DASD page data sets to keep the unreferenced pages, when there is a real storage frames shortage. These data sets are formatted in 4 KB slots.
- Supporting page faults, bringing to a real storage frame the copy of the page from the page data sets.
- Executing page stealing, when the amount of available frames falls below a certain threshold. The pages to be stolen are the least referenced. Pages in a central storage frame have an associated count called the **unreferenced-interval count** (UIC) that is calculated by MVS. This count measures for how many seconds such a page is unreferenced. The pages with the highest UIC counts are the ones to be stolen.

**Address space tables**

MVS creates a segment table for each address space. The segment table entries point, as shown in Figure 1-31 on page 59, to page tables for only the ones representing pages with valid data in them in the address space. If a 4 K block of virtual addresses has no valid data in it, there is no need to build a page table entry. There is a bit in segment table entry (the invalid bit) that indicates this condition to DAT.

DAT knows the current address space because MVS loads, in control register 1, the real address of its segment table. As shown in Figure 1-32 on page 60, the same virtual address 10254000 is referred to in two different address spaces that point to distinct real addresses. Each address has its own segment table.

z/OS implements a common area in all address spaces. All the segments belonging to the common area share the same set of page tables. Refer to *ABCs of z/OS System Programming Volume 2*, SG24-6982, for more information about the common areas.

If the virtual address is above the bar in the address space, the segment table is not used and the appropriate region table is used for DA: Third Region table, Second Region table and First Region table; refer to “z/Architecture enhancements” on page 12.
1.33 Dual address space (cross memory)

MVCP and MVCS
The PC routine can, if necessary, access data in the user's address space without using ARs. The MVCP instruction moves data from the secondary address space (the user) to the primary address space (the service provider). The MVCS instruction moves data from the primary address space (the service provider) to the secondary address space (the user). To use the MVCP or MVCS instructions, the service provider must have obtained SSAR authority to the user's address space before the PC routine receives control.

Cross memory
Synchronous cross-memory communication enables one program to provide services synchronously to other programs. Synchronous cross-memory communication takes place between address space 2, which gets control from address space 1 when the program call (PC) instruction is issued. Address space 1 has previously established the necessary environment, before the PC instruction transfers control to an address space 2 program called a PC routine. The PC routine provides the requested service and then returns control to address space 1.

The user program in address space 1 and the PC routine can execute in the same address space or, as shown in Figure 1-33, in different address spaces. In either case, the PC routine executes under the same TCB as the user program that issues the PC. Thus, the PC routine provides the service synchronously.
Dual address space or cross-memory (XM) is an evolution of virtual storage. It has three objectives:

- Move data synchronously between virtual addresses located in distinct address spaces.
  
  This can be implemented by the use of the SET SECONDARY ADDRESS REGISTER (SSAR) instruction. It points to an address space and makes it secondary. A secondary address space has its Segment Table pointed to by CR 7 instead of CR 1. Next, using MOVE CHARACTER TO SECONDARY (MVCS) or MOVE CHARACTER TO PRIMARY (MVCP), the objective can be accomplished.

- Pass the control synchronously between instructions located in distinct address spaces.
  
  There is an instruction, PROGRAM CALL (PC), able to do that. To return the origin address space, there is the instruction PROGRAM RETURN (PR).

- Execute one instruction located in one AS and its operands are located in another address space.
  
  Through the SSAR instruction, a secondary address space is defined. Using the SET ADDRESSABILITY CONTROL (SAC) instruction, the CP is placed in “Secondary Space Translation mode”. When in this mode, the instruction's virtual address is translated by DAT through CR1 and the operand's virtual addresses through CR7. Then, it is possible to have more than one set of active DAT tables (for instructions or operands) for translation depending on the translation mode as indicated by bits 16 and 17 in PSW, which are modified by the SAC instruction.
1.34 Access register mode (dataspaces)

Access registers (ARs)

An access register (AR) is a hardware register that a program uses to identify an address space or a data space. Each server has 16 ARs, numbered 0 through 15, and they are paired one-to-one with the 16 general purpose registers (GPRs).

Dataspaces and hiperspaces

Dataspaces and hiperspaces are data-only spaces that can hold up to 2 gigabytes of data. They provide integrity and isolation for the data they contain in much the same way as address spaces provide integrity and isolation for the code and data they contain. They are a very flexible solution to problems related to accessing large amounts of data.

There are two basic ways to place data in a data space or a hiperspace. One way is through using buffers in the program's address space. A second way avoids the use of address space virtual storage as an intermediate buffer area; instead, through data-in-virtual services, a program can move data into a data space or hiperspace directly. For hiperspaces, the second way reduces the amount of I/O.

Programs that use data spaces run in AR ASC mode. They use MVS macros to create, control, and delete data spaces. Assembler instructions executing in the address space directly manipulate data that resides in data spaces.
Using access registers

Access registers provide you with a different function from cross-memory. You cannot use them to branch into another address space. Through access registers, however, you can use assembler instructions to manipulate data in other address spaces and in data spaces. You do not use access registers to reference addresses in hiperspaces.

Through access registers your program, whether it is supervisor state or problem state, can use assembler instructions to perform basic data manipulation, such as:

- Compare data in one address space with data in another.
- Move data into and out of a data space, and within a data space.
- Access data in an address space that is not the primary address space.
- Move data from one address space to another.
- Perform arithmetic operations with values that are located in different address spaces or data spaces.

A PC routine can access (fetch or store) data in the user's address space by using access registers (ARs) and the full set of assembler instructions. If the PC routine has the proper authority, it can also access data in other address spaces or in dataspaces.

Access register mode is an extension of cross-memory. It introduces the concept of the dataspace. A dataspace is a sort of address space (but at the old 2 GB size) and consequently it is also represented by segment tables. The major difference is that an address space contains the addresses of instructions and operands, and a dataspace only contains operand addresses.

Dataspaces

In order to access a dataspace, the CP must be in access register translation mode, as set by the SAC instruction on bits 16 and 17 in the current PSW. In this mode, DAT translates instruction virtual addresses through the segment table pointed to by CR 1 (for address spaces) and operands through the segment table pointed by an access register. Refer to 1.10, “CP registers (general)” on page 20 for more detailed information. An access register points indirectly to the data space's segment table.

The advantage of having dataspaces is a cleaner design for programming, where data and instructions are not mixed.
1.35 CPU signaling facility

- 1 - Sense
- 2 - External call
- 3 - Emergency signal
- 4 - Start
- 5 - Stop
- 6 - Restart
- 9 - Stop and store status
- B - Initial CP reset
- C - CP reset
- D - Set prefix
- E - Store status at address
- 12 - Set architecture

Figure 1-35 Signaling facility order codes

Signaling facility
The CP signaling facility consists of a SIGNAL PROCESSOR (SIGP) instruction and a mechanism to interpret and act on several order codes. The facility provides for communication among CPs, including transmitting, receiving, and decoding a set of assigned order codes; initiating the specified operation; and responding to the signaling CP. This facility fulfills the MVS need for communication between its components, when running in different CPs in the same tightly coupled complex.

Signal-processor orders
Signal-processor orders are specified in bit positions 56-63 of the second-operand address of SIGNAL PROCESSOR instruction, and are encoded as shown in Figure 1-35.

Some of the orders are as follows:
- Change the CP architecture (12) from ESA/390 to z/Architecture, and vice versa.
- Generate external interrupts as such (2) and (3).
- Cause a stop and store the CP status in memory (9).
- Reset the CP (C). CP reset provides a means of clearing equipment check indications and any resultant unpredictability in the CP state, with the least amount of information destroyed.
1.36 Time measurement TOD

The timing facilities include three facilities for measuring time:

- The TOD clock
- The clock comparator
- The CP timer

A TOD programmable register is associated with the TOD clock. In a multiprocessing configuration, a single TOD clock is shared by all CPs. Each CP has its own clock comparator, CP timer, and TOD programmable register.

The TOD clock is a 104-bit counter register inside each PU. The TOD clock provides a high resolution measure of real time suitable for the indication of date and time of day. This timing is precious information for operating systems, databases, and system logs.

The cycle of the clock is approximately 143 years (from all bits zero to all bits zero again). The TOD clock nominally is incremented by adding a one in bit 51 every microsecond. In models having a higher or lower resolution, a different bit position is incremented at such a frequency that the rate of advancing the clock is the same as if a one were added in bit 51 every microsecond.

TOD follows the coordinated universal time (UTC) that is derived from the atomic time TA1 (based in Cesium 133 radioactivity), and is adjusted with discrete leap seconds to keep reasonably close to UT1 (based on the Earth’s rotation).
Incrementing the TOD clock does not depend on whether the CP is in a wait state or whether the CP is in operating, load, stopped, or check-stop states.

**Note:** The TOD cannot be used by MVS for time accounting for tasks in z/OS.

**Instructions for storing the clock**

Two instructions are used by MVS to alter and store its contents in memory:

- The SET CLOCK EXTENDED instruction changes the contents of 104 bits TOD from a memory location.
- The STORE CLOCK EXTENDED instruction, as pictured in Figure 1-36 on page 67, moves into memory 104 bits plus the program fields (16 bits) from the TOD program fields register.

MVS assumes that 01/01/1900 corresponds to a TOD zeroed. The *clock comparator* is a circuit in each PU that provides an external interrupt (X'1004') when the TOD clock value exceeds a value specified by the program. Using the clock comparator, the software can be alerted when a certain amount of wall clock time has elapsed, or at a specific hour of the day.
1.37 Time measurement (CP timer)

- Decrementing binary counter register in the CP
- 64 bits (TOD vs ETOD)
- Same frequency as TOD clock
- Stops when the CP is not running or not in wait
- Used for CP accounting purposes by MVS
- Causes external interrupt when reaching a negative value
- Instructions: STORE CPU TIMER and SET CPU Timer

**Figure 1-37 CP timer**

**CP timer**
The CP timer is a binary counter with a format which is the same as that of bits 0-63 of the TOD clock, except that bit 0 is considered a sign. The CP timer nominally is decremented by subtracting a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is decremented at such a frequency that the rate of decrementing the CP timer is the same as if a one were subtracted in bit position 51 every microsecond. The CP timer requests an external interrupt with the interrupt code 1005 hex whenever the CP timer value is negative (bit 0 of the CP timer is one).

**TOD and ETOD formats**
It is recommended that you begin to convert your applications to using the ETOD format. The extended time-of-day format was required both to address the time-wrapping problem that would occur in the year 2042, and also to provide the improved resolution necessary for faster servers as they become available.

**Note:** If you request ETOD information and your server is not configured with the 128-bit extended time-of-day clock, timer services will return the contents of the 64-bit TOD and simulate the remaining 64 bits of the ETOD.

Conversely, if you request TOD information and your server is configured with the extended time-of-day clock, timer services will return only that portion of the 128-bit ETOD that corresponds to the 64-bit TOD.
**TOD frequency**
When both the CP timer and the TOD clock are running, the stepping rates are synchronized such that both are stepped at the same rate when a specified amount of time has elapsed.

**Timer stops**
The CP timer stops when the CP is in stop state. This state may be caused by operator intervention, or in LPAR mode when a shared logical CP is not executed in a physical CP.

**MVS and accounting**
The CP timer is used by MVS for accounting purposes (because it stops at CP stop); that is, it is used to time how much CP a task or a service request consumes.

**Timer external interrupt**
Assume that a program being timed by the CP timer is interrupted for a cause other than the CP timer, external interruptions are disallowed by the new PSW, and the CP timer value is then saved by STORE CPU TIMER. This value could be negative if the CP timer went from positive to negative since the interruption.

Subsequently, when the program being timed is to continue, the CP timer may be set to the saved value by SET CPU TIMER. A CP timer interruption occurs immediately after external interruptions are again enabled if the saved value was negative.

**Timer instructions**
The CP timer can be inspected by executing the instruction STORE CPU TIMER, and can be set by MVS to a specified value by executing the SET CPU TIMER instruction.
1.38 Sysplex Timer expanded availability configuration

![Diagram of Sysplex Timer expanded availability configuration]

External timer reference (ETR)

There is a long-standing requirement for accurate time and date information in data processing. As single operating systems have been replaced by multiple, coupled operating systems on multiple servers, this need has evolved into a requirement for both accurate and consistent clocks among these systems. Clocks are said to be “consistent” when the difference or offset between them is sufficiently small. An accurate clock is consistent with a standard time source.

The IBM z/Architecture and S/390 Architecture external time reference (ETR) architecture facilitates the synchronization of server time-of-day (TOD) clocks to ensure consistent time stamp data across multiple servers and operating systems. The ETR architecture provides a means of synchronizing TOD clocks in different servers with a centralized time reference, which in turn may be set accurately on the basis of an international time standard (External Time Source). The architecture defines a time-signal protocol and a distribution network, called the ETR network, that permits accurate setting, maintenance, and consistency of TOD clocks.

ETR time

In defining an architecture to meet z/Architecture and S/390 Architecture time coordination requirements, it was necessary to introduce a new kind of time, sometimes called ETR time, that reflects the evolution of international time standards, yet remains consistent with the original TOD definition. Until the advent of the ETR architecture (September 1990), the server TOD clock value had been entered manually, and the occurrence of leap seconds had been
essentially ignored. Introduction of the ETR architecture has provided a means whereby TOD clocks can be set and stepped very accurately, on the basis of an external Universal Time Coordinate (UTC) time source.

**Sysplex Timer**
The IBM 9037 Sysplex Timer is a mandatory hardware requirement for a Parallel Sysplex consisting of more than one zSeries server. The Sysplex Timer provides the synchronization for the time-of-day (TOD) clocks of multiple servers, and thereby allows events started by different servers to be properly sequenced in time. When multiple servers update the same database, all updates are required to be time stamped in proper sequence.

Sysplex Timer model 002 supports two types of configuration:
- **Basic configuration**
- **Expanded availability configuration**

Sysplex Timer Expanded Availability configuration is the recommended configuration in a Parallel Sysplex environment. This configuration is fault-tolerant to single points of failure, and minimizes the possibility that a failure can cause a loss of time synchronization information to the attached servers.

**Expanded availability configuration**
The Sysplex Timer expanded availability configuration consists of two IBM 9037 Sysplex Timer Units, as shown in Figure 1-38 on page 71. In an Expanded Availability configuration, the clocks running in the two Sysplex Timer units are synchronized to each other using the control link oscillator (CLO) card in each Sysplex Timer unit and the CLO links between them.

Both Sysplex Timer units are simultaneously transmitting the same time synchronization information to all attached servers. The CLO connection between Sysplex Timer units is duplicated, to provide redundancy. Critical information is exchanged between the two Sysplex Timer units, so in case one fails, the other unit will continue transmitting to the attached servers without disrupting server operations.

Redundant fiber optic cables are used to connect one port from each Sysplex Timer unit to the two ETR ports of a server. Each attaching server's two ETR ports (ETR Port 0 and ETR Port 1) operate in one of two modes: the active (or stepping) ETR port, and the alternate ETR port. The server's TOD clock steps to the timing signals received from the active (stepping) ETR port only. If a server's alternate ETR port detects the stepping ETR port to be nonoperational, it forces an automatic ETR port switchover; the server's TOD clock now steps to timing signals received from the server's alternate ETR port, which has now assumed the role as the stepping ETR port. This switchover takes place without disrupting server operations.

**Note:** The Sysplex Timer units do not switch over, and are unaware of the ETR port role change at the server end.

For an effective fault-tolerant Sysplex Timer Expanded Availability configuration, ETR Port 0 and ETR Port 1 in each server must connect to different Sysplex Timer units within the same Sysplex Timer ETR network.
1.39 Server Time Protocol (STP)

Server Time Protocol (STP) is designed for z/OS V1R7 (PTFs are required) running on a z196 or z10 EC, z9, z990, or z890 that requires the time of day (TOD) clock to be synchronized. STP is a replacement for the Sysplex Timer, and it is aimed at decreasing the total cost of the mainframe platform. STP allows TOD information to be exchanged between z/OS systems running in different servers using CF links as communication media. STP is designed to:

- Support a multisite timing network of up to 100 km (62 miles) over fiber optic cabling, allowing a Parallel Sysplex to span these distances
- Potentially reduce the cross-site connectivity required for a multisite Parallel Sysplex
- Coexist with an ETR (Sysplex Timer) network
- Allow use of dial-out time services to set the time to an international time standard (such as coordinated universal time (UTC), for example), as well as adjust to UTC on a periodic basis
- Allow setting of local time parameters, such as time zone and daylight saving time
- Allow automatic updates of daylight saving time

All the functions performed previously through the Sysplex Timer console are executed in the server HMC. Stratum 1 is the z/OS propagating TOD signals to stratum 2 z/OS systems. Stratum 2 receives from stratum 1 and propagates to stratum 3. Stratum 0 is the external time source (ETS).
1.40 Data center and I/O configuration

You must define an I/O configuration to the operating system (software) and the channel subsystem (hardware). The Hardware Configuration Definition (HCD) element of z/OS consolidates the hardware and software I/O configuration processes under a single interactive end-user interface. The validation checking that HCD does as you enter data helps to eliminate errors before you attempt to use the I/O configuration.

An I/O configuration is the hardware resources available to the operating system and the connections between these resources. The resources include:

- Channels
- ESCON/IBM FICON® Directors (switches)
- Control units
- Devices such as tape, printers, and DASD

Figure 1-40 shows a typical zSeries data center. As you can see, the complex consists of separate I/O devices and networks connected through high-speed data buses to the server, which comprises servers, server storage, and channels. It shows connections among servers, as well. z/Architecture provides up to 512 high-speed data buses, called channels, per server. Included in those are the OSA channels, which are assembled with a network controller and one adapter.
Channel types
Input/Output (I/O) channels are components of the zSeries and 9672 G5/G6 Channel Subsystems (CSS). They provide a pipeline through which data is exchanged between servers, or between a server and external devices. The different types of channels, including the ones connecting the Coupling Facility (CF) are:

- **Parallel channels** - IBM introduced the parallel channel with System/360 in 1964. The I/O devices were connected using two copper cables called bus cables and tag cables. A bus cable carries information (one byte each way), and a tag cable indicates the meaning of the data on the bus cable.

- **Enterprise Systems Connection (IBM ESCON®)** - Since 1990, ESCON has replaced the parallel channel as being the main channel protocol for I/O connectivity, using fiber optic cables and a new “switched” technology for data traffic.

- **Fiber Connection: FICON, FICON Express and Fibre Channel Protocol (FCP)** - In 1998, IBM announced a new I/O architecture for 9672 G5/G6 servers called Fibre Connection (FICON). The zSeries server builds on this architecture by offering higher speed FICON connectivity.

- **Open Systems Adapter-2 (OSA-2)**, with the following type of controllers: Ethernet, Fast Ethernet, Token-ring, Fiber Distributed Data Interface (FDDI), 155 Asynchronous Transfer Mode (ATM) - In 1995, OSA-2 was introduced, bringing the strengths of zSeries, such as security, availability, enterprise-wide access to data, and systems management to the client/server environment. Some OSA-2 features continue to be supported in z/Architecture (z900 only), while others have been replaced by the OSA-Express features, which were introduced in 1999. All OSA features were designed to provide direct, industry standard, local area network (LAN) and ATM network connectivity in a multi-vendor networking infrastructure.

- **OSA Express**, with the following type of controllers: Gigabit Ethernet, Fast Ethernet, 1000BaseT Ethernet, High Speed Token Ring) - The Open Systems Adapter-Express (OSA-Express) Gigabit Ethernet (GbE), 1000BASE-T Ethernet, Fast Ethernet (FENET), Asynchronous Transfer Mode (ATM) and Token Ring (TR) features are the generation of features beyond OSA-2. OSA-Express features provide significant enhancements over OSA-2 in function, connectivity, bandwidth, data throughput, network availability, reliability, and recovery.

- **ISC** - fiber Coupling Facility (CF) link - InterSystem Channels (ISC) provide the connectivity required for data sharing between the CF and the systems directly attached to it. ISC links are point-to-point connections that require a unique channel definition at each end of the link.

- **ICB** - copper Coupling Facility (CF) link - Integrated Cluster Bus links are members of the Coupling Link options available on zSeries and G5/G6 servers. They are faster than ISC links, attaching directly to a Self-Timed Interconnect (STI) bus of the server.

Data transfer mechanisms
Methods of transferring data are as follows:

- **Traditional I/O data transfer** through the use of the START SUBCHANNEL instruction, CCWs, and I/O interrupts, used by ESCON and FICON channels.

- **Queue Direct I/O (QDIO)**, through the use of the SIGA instruction, used by OSA-Express and IBM HiperSockets™ channels. QDIO is a highly efficient data transfer mechanism that satisfies the increasing volume of TCP/IP applications and increasing bandwidth demands. It dramatically reduces system overhead, and improves throughput by using system memory queues and a signaling protocol to directly exchange data between the OSA-Express microprocessor and TCP/IP software.

- **Message architecture**, to access the Coupling Facility (CF), used by CF links (ISC, ICB, IC).
1.41 Channel subsystem (CSS) elements

Channel Subsystem is formed by:

- Up to 256 channels
- A few SAPs (depends on the zSeries model)
- Up to 64 K subchannels (also called UCWs)
- I/O devices attached through control units

Channel Path Identifier (CHPID)

Control units

I/O devices

Channel subsystem

A subchannel provides the logical representation of a device to the program and contains the information required for sustaining a single I/O operation. A subchannel is assigned for each device defined to the logical partition. Three subchannel sets are available to z196. Two subchannel sets are available on z114 and System z10®.

An individual I/O device may be accessible to the channel subsystem by as many as eight different channel paths, depending on the model and the configuration. The total number of channel paths provided by a channel subsystem depends on the model and the configuration; the maximum addressability is 256.

Channels

The zSeries channel subsystem contains channels, which are much simpler than an SAP, are able to communicate with I/O control units (CU) and manage the movement of data between central storage and the control units. They are located in I/O cards in the zSeries I/O cage; refer to “I/O cages” on page 120. Being more specific, the channels can:

- Send channel commands from the memory to a CU
- Transfer data during read and write operations
- Receive status at the end of operations
- Receive sense information from control units, such as detailed error information
In z/Architecture, the I/O operation is created when a privileged START SUBCHANNEL (SSCH) instruction is executed by the input/output supervisor (IOS), a z/OS component, which issues the instruction on behalf of a task. It finishes when an I/O interrupt is received by the CPU, forcing the execution of the IOS component again.

**SAPs**

In order to communicate with the I/O devices attached to the channels, both the operating system (MVS) and the channel subsystem (formed by the system-assisted processor (SAP) and channels) need to know the I/O configuration, which contains the following types of information:

- What devices are attached to each channel
- The device types
- The device addresses: subchannel number, device number, and device address (also called a unit address)
- What protocols to use
- Channel type

A PU is a zSeries server. An SAP is a PU that runs I/O licensed internal code (LIC). By LIC, IBM means either microcode or software programs that the customer is not able to read or alter. The SAP relieves z/OS (and consequently, CP involvement) during the setup of an I/O operation. It does the scheduling of an I/O operation; that is, it finds an available channel path to the device and guarantees that the I/O operation really starts. SAP, however, is not in charge of the movement between central storage (CS) and the channel.

**Subchannels**

Within the channel subsystem are subchannels. One subchannel is provided for and dedicated to each I/O device accessible to the channel subsystem. Each subchannel provides information concerning the associated I/O device and its attachment to the channel subsystem. The subchannel also provides information concerning I/O operations and other functions involving the associated I/O device.

The subchannel is the means by which the channel subsystem provides information about associated I/O devices to CPs, which obtain this information by executing I/O instructions. The actual number of subchannels provided depends on the model and the configuration; the maximum addressability is 65,536.

**Control units**

A control unit provides the logical capabilities necessary to operate and control an I/O device and adapts the characteristics of each device so that it can respond to the standard form of control provided by the CSS. A control unit can be housed separately, or it can be physically and logically integrated with the I/O device, the channel subsystem, or within the server itself.

**I/O devices**

An I/O device provides external storage, a means of communication between data-processing systems, or a means of communication between a system and its environment. In the simplest case, an I/O device is attached to one control unit and is accessible through one channel path.
1.42 Multiple CSS structure (z10 and z196 EC)

<table>
<thead>
<tr>
<th>LPARs Names</th>
<th>LP1</th>
<th>LP2</th>
<th>LP3</th>
<th>LP14</th>
<th>LP15</th>
<th>LP16</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIF IDs</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>LCSSs</td>
<td>LCSS 0</td>
<td></td>
<td></td>
<td>LCSS 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHPIDs</td>
<td>80</td>
<td>81</td>
<td>90</td>
<td>91</td>
<td>80</td>
<td>81</td>
</tr>
<tr>
<td>PCHIDs</td>
<td>140</td>
<td>150</td>
<td>1E0</td>
<td>1F0</td>
<td>141</td>
<td>151</td>
</tr>
</tbody>
</table>

Figure 1-42  Multiple CSS structure for the z10 and z196 EC

Multiple CSS - description

The CSS architecture provides functionality in the form of multiple CSSs. Multiple CSSs can be configured within the same System z (up to four on z196 and z10 EC, and up to two on z114 and z10 BC servers. This delivers a significant increase in I/O scalability.

System z delivers a considerable increase in system performance due to an improved design, use of technology advances, and decrease in cycle time. In support of this, the CSS is correspondingly scaled to provide for significantly more channels. New concepts are introduced to facilitate this architectural change and provide relief for the number of supported LPARs, channels, and devices available to the server. The structure of multiple CSSs provides channel connectivity to the defined logical partitions in a manner that is transparent to subsystems and application programs.

The System z servers provide the ability to define more than 256 CHPIDs in the system through the multiple CSSs. As mentioned, CSS defines CHPIDs, control units, subchannels, and so on. This enables the definition of a balanced configuration for the processor and I/O capabilities.

For ease of management, consider using the Hardware Configuration Definitions (HCD) tool to build and control the System z input/output configuration definitions. HCD support for multiple channel subsystems is available with z/VM and z/OS. HCD provides the capability to make both dynamic hardware and software I/O configuration changes.
Logical partitions cannot be added until at least one CSS has been defined. Logical partitions are defined to a CSS, not to a server. A logical partition is associated with one CSS only. CHPID numbers are unique within a CSS. However, the same CHPID number can be reused within all CSSs.

**Important:** The System z servers do not support basic mode. Only logically partitioned (LPAR) mode can be defined.

**Multiple CSS - components**
The multiple CSS introduces new components and terminology that differs from previous server generations. These components are explained in the following sections.

**Important:** A zArchitecture server does not support basic mode. Only LPAR mode can be defined.

**Channel subsystem (CSS)**
Beginning with the z990, a server provides the ability to have more than 256 CHPIDs in the system by the introduction of the multiple channel subsystem. This is a logical replication of CSS facilities (CHPIDs, control units, subchannels, and so on). This enables a balanced system structure between the server's capability and the I/O capability. The CSS for the z10 EC server introduces significant changes to the I/O configuration. For ease of management, it is strongly recommended that HCD is used to build and control your z10 and z196 EC Input/Output configuration definitions.

It is important to note that the z10 EC is still a single server with logical extensions. All Channel Subsystem Images (CSS Image or CSS) are defined within a single IOCDS. The IOCDS is loaded into the Hardware System Area (HSA) and initialized during Power-on Reset by the Support Element.

A CSS is identified by a CSS Identifier (CSS ID). The CSS IDs are 0 (zero), 1, 2, and 3.

**Multiple channel subsystem**
The multiple channel subsystem (CSS) concept implemented in the System z servers is designed to offer a considerable increase in processing power, memory sizes, and I/O connectivity over previous servers; refer to Table 1-1 for a comparison.

<table>
<thead>
<tr>
<th></th>
<th>z10 BC</th>
<th>z10 EC</th>
<th>z114</th>
<th>z196</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CSSs</strong></td>
<td>2 per server</td>
<td>4 per server</td>
<td>2 per server</td>
<td>4 per server</td>
</tr>
<tr>
<td><strong>Partitions</strong></td>
<td>15 per CSS, up to 30 per server</td>
<td>15 per CSS, up to 60 per server</td>
<td>15 per CSS, up to 30 per server</td>
<td>15 per CSS, up to 60 per server</td>
</tr>
<tr>
<td><strong>CHPIDs</strong></td>
<td>256 per CSS, up to 512 per server</td>
<td>256 per CSS, up to 1024 per server</td>
<td>256 per CSS, up to 512 per server</td>
<td>256 per CSS, up to 1024 per server</td>
</tr>
<tr>
<td><strong>Devices</strong></td>
<td>63.75 K per CSS&lt;sup&gt;a&lt;/sup&gt;, up to 255 K per server</td>
<td>63.75 K per CSS&lt;sup&gt;a&lt;/sup&gt;, up to 255 K per server</td>
<td>63.75 K per CSS&lt;sup&gt;a&lt;/sup&gt;, up to 255 K per server</td>
<td>63.75 K per CSS&lt;sup&gt;a&lt;/sup&gt;, up to 255 K per server</td>
</tr>
</tbody>
</table>

<sup>a</sup> Multiple subchannel sets are supported.
1.43 Control units

- Every channel connects to a physical control unit through a host adapter
- The same physical control unit can play the role of several logical control units
- The control unit may be located:
  - In a separate unit with several devices (as ESS)
  - In the CEC, such as CTCA or OSA
  - In the device (as tape)

Control units
A control unit provides the logical capabilities necessary to operate and control an I/O device, and adapts the characteristics of each device so that it can respond to the standard form of control provided by the channel subsystem.

Communication between the control unit and the channel subsystem takes place over a channel path. The control unit accepts control signals from the channel subsystem, controls the timing of data transfer over the channel path, and provides indications concerning the status of the device.

The I/O device attached to the control unit may be designed to perform only certain, limited operations, or it may perform many different operations. A typical operation is moving a recording medium and recording data. To accomplish its operations, the device needs detailed signal sequences peculiar to its type of device. The control unit decodes the commands received from the channel subsystem, interprets them for the particular type of device, and provides the signal sequence required for the performance of the operation.

Control unit functions
All channels connect to a control unit prior to connecting to the I/O device. The role of the control unit is to regulate the I/O-device operation. It provides intelligence, caching and buffering capabilities necessary to operate multiple I/O requests to the associated I/O devices. It also does error recovery. Today's control units are very complex, behaving as several independent logical control units; they are sometimes called I/O subsystems.
From the z/OS and application programming perspective, most control unit functions merge with I/O device functions.

The control unit function may be:

- In a separate control unit
- Integrated with the I/O device, as for some tape devices
- Integrated with the channel in the server as CTCA, OSA

**I/O devices**

An input/output (I/O) device is the endpoint in the “conduit” between a server and a peripheral. Although the channel does not communicate directly with I/O devices (it communicates with control units), it is useful to mention them here because we previously discussed subchannels, which appear as I/O devices to programs.
1.44 Device number

Each subchannel that has an I/O device assigned to it also contains a system-unique parameter called the **device number**. The device number is a 16-bit value that is assigned as one of the parameters of the subchannel at the time the device is assigned to the subchannel.

The device number provides a means to identify a device, independent of any limitations imposed by the system model, the configuration, or channel-path protocols. The device number is used in communications concerning the device that take place between the system and the system operator. For example, the device number is entered by the system operator to designate the input device to be used for initial program loading (IPL), or the operator once the IPL completes can vary a device online or offline, as follows:

V 200A, online

**HCD**

A device number is a nickname you assign to identify all the I/O devices in the configuration, using HCD. A device number may be any hexadecimal number from X'0000' to X'FFFF' allowing a maximum of 65,536 devices. You should specify the physical unit address in the Unit Address field in the HCD panel.

**HSA**

The device number is stored in the UCW (in the Hardware System Area - HSA) at Power-on Reset (POR) and comes from the IOCDS. HSA is a piece of central storage not addressable.
by z/OS. It is allocated at Power-on Reset (POR) and contains LPAR LIC, microcode work areas, and the I/O configuration as UCWs used by the channel subsystem. For each pair device/logical partition, there is one UCW representing the device in HSA.

**UCB**
The device number is also stored in the UCB at IPL, to be used by IOS. A unit control block (UCB) holds information about an I/O device, such as:

- State information for the device
- Features of the device

The UCW represents the device from the channel subsystem point of view; an UCB represents the device from the IOS point of view.

**Device number considerations**
If your installation is running out of device numbers, you might use the same device number for two different devices across several z/OS LPARs. The devices have to be attached to different control units. To indicate whether devices identified by the same device number are the same or a different device, you can specify the serial number of the device to HCD for documentation purposes only.

Do not name the same device with different device numbers in different z/OS LPARs, as this may cause some confusion for your operating staff. However, z/OS perceives that the device is the same, because it can recognize the device by a CCW called read device characteristics.
1.45 Subchannel number

- Device identification used for communication
  - Between CP and channel subsystem

- Two bytes
  - Values from 0000 to FFFF (64K devices)

- Index for the UCW in the UCW table

- Used by SSCH microcode
  - To find the requested UCW

- Stored in the UCB during the IPL process

Subchannel number

Normally, subchannel numbers are only used in communication between the CP program and the channel subsystem.

A subchannel number is a system-unique 16-bit (2 bytes) value whose valid range is 0000-FFFF and which is used to address a subchannel. The subchannel is addressed by eight different I/O instructions. Each I/O device accessible to the channel subsystem is assigned a dedicated subchannel at installation time.

All I/O functions relative to a specific I/O device are specified by the program by designating the subchannel assigned to the I/O device. Subchannels are always assigned subchannel numbers within a single range of contiguous numbers.

The lowest-numbered subchannel is subchannel 0. The highest-numbered subchannel of the channel subsystem has a subchannel number equal to one less than the number of subchannels provided. A maximum of 65,536 subchannels, (64 K) devices per LPAR per CSS, can be provided.

A subchannel is a z/Architecture entity. It provides the logical appearance of a device to z/OS, and contains information required for sustaining a single I/O operation. I/O operations are initiated with a device by the execution of I/O instructions that designate the subchannel associated with the device.
In zSeries servers, subchannels are implemented through UCWs, control blocks allocated in the Hardware System Area (HSA) at Power-on Reset (POR). Then, the subchannel consists of internal storage (UCW) that contains two types of information:

- **Static** from the HCD process, describing the paths to the device (CHPIDs), device number, I/O-interrupt-subclass code, as well as information on path availability.
- **Dynamic** referred to ongoing I/O operation, such as functions pending or being performed, next CCW address, status indication. There is just one I/O operation per UCW.

The subchannel number is stored in the UCB at IPL.

**Note:** When you have devices in control units with the capability of doing parallel I/O operations (Parallel Access Volume - PAV) as Shark, you need to define a UCW for each additional parallel I/O.
1.46 Subchannel numbering

Subchannel numbering

Subchannel numbers are limited to four hexadecimal digits by hardware and software architectures. Four hexadecimal digits provides up to 64 K addresses, known as a set. IBM reserved 256 subchannels, leaving 63.75 K subchannels for general use.

The advent of Parallel Access to Volumes (PAV) has made this 63.75 K subchannels limitation a problem for larger installations. One solution is to have multiple sets of subchannels, with a current implementation of two sets. Each set provides 64 K-1 addresses. Subchannel set 0 still reserves 256 subchannels for IBM use. Subchannel set 1 provides to the installation the full range of 64 K addresses.

Figure 1-46 shows the offset concept of the subchannel number in the UCW table as assigned during installation. When IOCP generates a basic IOCDS, it creates one subchannel for each I/O device associated with a logical control unit.

When IOCP generates an LPAR IOCDS, the number of subchannels it generates for the group of I/O devices associated with a logical control unit depends on the number of logical control units generated for the group of I/O devices. The total number of subchannels it generates for the group of I/O devices is the number of I/O devices in the group multiplied by the number of logical control units generated for that group.

The device may be a physically identifiable unit, or it may be housed internal to a control unit. In all cases a device, from the point of view of the channel subsystem, is an entity that is uniquely associated with one subchannel and that responds to selection by the channel.
subsystem by using the communication protocols defined for the type of channel path by which it is accessible.

Then, the subchannel number is another way for addressing an I/O device. It is a value that is assigned to the subchannel (UCW) representing the device at POR time. It indicates the relative position of the UCW in the UCW table. The subchannel number was designed to speed up the search of a UCW during SSCH processing.

In a z990, the maximum number of UCWs is 64 K times 15 times 2, because there is one UCW table per LPAR per CSS in the HSA; refer to “Logical channel subsystem (LCSS)” on page 249. Then the same subchannel number is duplicated in different CSSs. However, it does not pose a problem because the same z/OS can be in just one CSS.
1.47 Control unit address

Control units

A control unit provides the logical capabilities necessary to operate and control an I/O device, and adapts the characteristics of each device so that it can respond to the standard form of control provided by the channel subsystem.

Communication between the control unit and the channel subsystem takes place over a channel path. The control unit accepts control signals from the channel subsystem, controls the timing of data transfer over the channel path, and provides indications concerning the status of the device.

Modern DASD I/O control units may offer a great deal of space, already in the terabyte realm. To reach these figures, they need to have thousands of I/O devices. However, z/Architecture only allows 256 devices, per channel per I/O control unit, as we see in 1.48, “Unit addresses” on page 90.

To circumvent this situation, DASD manufacturers use a technique created by an I/O architect where, for the 3174 network controllers, the same physical control unit responds for several logical control units, as shown in Figure 1-47. Then, each DASD control unit responds to the channel as several logical control units (each one with up to 256 devices), each identified by a CUA as indicated in the header of the frames used in the communication between channels and I/O control units.

ESCON channels supports up to 16 I/O logical control units in the CUA (4 bits) field and FICON supports up to 256 I/O logical control units (8 bits). Then, the actual number of logical
control units in a physical I/O control unit depends on the model and the manufacturers of each one.

**ESCON Directors**

Figure 1-47 on page 88 shows ESCON Directors (ESCD). The switching function of ESCON is handled by products called ESCON Directors, which operationally connect channels and control units only for the duration of an I/O operation (dynamic connection). They can switch millions of connections per second, and are the centerpiece of the ESCON topology.

Apart from dynamic switching, the ESCON Directors can also be used for static switching of “single user” control units among different system images (dedicated connection).

**Note:** ESCON is an integral part of the Enterprise Systems Architecture/390 (ESA/390) and, by extension, of the z/Architecture. ESCON replaces the previous S/370 parallel OEMI with the ESCON I/O interface, supporting additional media and interface protocols.

By replacing the previous bus and tag cables and their multiple data and control lines, ESCON provides half-duplex serial bit transmission, in which the communication protocol is implemented through sequences of special control characters and through formatted frames of characters.
1.48 Unit addresses

- Two hex digits in range 00-FF
- Used to select a device on a channel
- Defined by a sysprog in HCD, taken from physical definition in the controller

Unit address

The unit address (UA) or device address is used to reference the device during the communication between a channel and the control unit serving the device. The UA is two-hex digits in the range 00-to-FF, and is stored in the UCW as defined to HCD. It is transmitted over the I/O interface to the control unit by the channel.

ESCON channel

An ESCON channel can reach up to 16 control units (the CUA field has 4 bits), with 256 devices each (the UA has 8 bits). An ESCON channel executes commands presented by the standard z/Architecture or ESA/390 I/O command set, and it manages its associated link interface (link level/device level) to control bit transmission and reception over the optical medium (physical layer). On a write command, the channel retrieves data from central storage, encodes it, and packages it into device-level frames before sending it on the link. On a read command, the channel performs the opposite operation.

Control unit

The UA has no relationship to the device number or the subchannel number for a device. Although Figure 1-49 on page 92 shows the device numbers at the control unit, the control unit is not aware of device numbers. Some control units attached to zSeries require a unit address range starting at X’00’, as shown in Figure 1-49 on page 92. The unit address defined in HCD must match the unit address on the control unit where the device has been physically installed by the hardware service representative.
I/O operation
In the visual, IOS issues the SSCH instruction pointing to the UCW with the subchannel number X’1E38’. One of the channels (24, for example) handling the I/O operation fetches from the UCW the UA 04. Channel 24 starts the dialogue with the control unit passing the UA 04, identifying the specific device for the I/O operation.

In summary:

- The device number is used in communication between an operator and MVS.
- The subchannel number is used in communication between CPU (z/OS) and channel subsystem.
- The unit address is used in communication between the channel and the control unit.
1.49 Map device number to device address

Figure 1-49 shows how, in an ESCON environment, the device number is mapped to the device address. The UNIT parameter in the DD card in the JCL specifies the device number through an esoteric or generic name. Also, a specific device number may be specified. This information is used to allocate the data set associated with this DD statement.

**Device address through JCL**

Figure 1-49 shows how, in an ESCON environment, the device number is mapped to the device address. The UNIT parameter in the DD card in the JCL specifies the device number through an esoteric or generic name. Also, a specific device number may be specified. This information is used to allocate the data set associated with this DD statement.

**MVS allocation**

Allocation of a data set in MVS terms means to associate the data set DD statement (through a TIOT, a control block) with the UCB of the device containing the data set. The UCB has all the information needed for the preparation of the SSCH instruction by IOS, including the subchannel number.

The channel subsystem, through a ESCON channel, finds in the UCW (through the subchannel number) the receiver address (port address in the ESCD switch, control unit, and unit address) to be placed in the ESCON frame.

Figure 1-49 also shows that any operator command that refers to a device uses the device number.
Multiple channel paths to a device

I/O devices

I/O devices are attached through I/O control units to the channel subsystem. The idea here is that, instead of having intelligence in all devices, the I/O designer centralized it in the I/O control unit. A channel path is simply a route to a device from the server.

An I/O device may be accessed by an z/OS system through as many as eight different channels. Also, an I/O device may be accessible to a server (channel subsystem) by as many as eight different channels.

Control units

Control units may be attached to the channel subsystem through more than one channel path. SAP is in charge of selecting the channel to reach the device; usually it balances the channel utilization by rotating the one selected. Also, an I/O device may be attached to more than one control unit.

DEVSEERV command

Use the DEVSEERV command to request a display of the status of DASD and tape devices. The response is a display of basic status information about a device, a group of devices, or storage control units, and optionally can include a broad range of additional information.
In Figure 1-50, the DEVSERV (DS) command output indicates that the device with a device number 0CC0 has 4 connected channels: 10, B0, 88 and DC. It also shows that CHPID DC is physically unavailable.

**DISPLAY M command**

Use the **DISPLAY M** command to display the status of sides, servers, vector facilities, ICRFs, channel paths, devices, central storage, and expanded storage, or to compare the current hardware configuration to the configuration in a CONFIGxx parmlib member. The system is to display the number of online channel paths to devices or a single channel path to a single device.

In Figure 1-50 on page 93, the command output shows the device status as online and for each of the CHPIDs, (10, B0, 88, and DC), whether the channel path in online, the CHPID is online, and the path is operational.

**Note:** The total number of channel paths provided by a channel subsystem depends on the model and the configuration; the maximum number of channel paths is 256 per server.
1.51 Start subchannel (SSCH) logic

START SUBCHANNEL (SSCH) instruction

SSCH is a privileged instruction issued by the Input/Output Supervisor (IOS) component of z/OS. IOS issues this instruction when responding to an I/O request from a task when there is not another ongoing I/O operation running in the device represented by the target UCB. If there is, this request is queued at the UCB level, also called the IOS queue. Remember that if it is a PAV device, then the I/O operation may start immediately.

The SSCH instruction has two operands:

- General register 1 contains a subsystem-identification word that designates the subchannel number to be started.
- The second-operand address is the logical address of the operation request block (ORB) and must be designated on a word boundary; otherwise, a specification exception is recognized.

Subchannel number

This is an index into the UCW associated with the I/O device (refer to 1.45, “Subchannel number” on page 84). It indicates the device where the I/O operation will be executed.

ORB

The ORB is an architected control block informing about what to do during the I/O operation; among other fields, it contains the channel program address. Refer to z/Architecture Reference Summary, SA22-7871, for the ORB contents.
SSCH logic
The channel subsystem is signaled to asynchronously perform the start function for the associated device, and the execution parameters that are contained in the designated ORB are placed at the designated subchannel, as follows:

- The SSCH microcode in the CP moves the ORB contents into the dynamic part of the respective UCW and places the UCW in a specific Hardware System Area (HSA) queue called the *initiative queue*.
  
  There is one initiative queue per SAP.

- The number of SAPs is server model-dependent, but in the z10 EC, you may have up to three SAPs per book; refer to “z10 EC Books” on page 123 for more detailed information. One I/O card is served by just one SAP, so depending on which channels serve the device, the right SAP is chosen by the SSCH microcode.

- After that SSCH logic completes, the next IOS instruction is executed, which later will allow the use of the CP in another task.
1.52 SAP PU logic

The SAP acts as an offload engine for the CPUs. Different server models have different numbers of SAPs, and a spare 9672 PU can be configured as an additional SAP.

SAP functions include:

- Execution of ESA/390 I/O operations. The SAP (or SAPs) are part of the I/O subsystem of the server and act as Integrated Offload Processor (IOP) engines for the other servers.
- Machine check handling and reset control.
- Support functions for the Service Call Logical Processor (SCLP).

The SAP finds the UCW in the initiative queue and tries to find an available channel path (channel, port switch, I/O control unit and device) that succeeds in starting the I/O operation. SAP uses the I/O configuration information described in the Hardware Configuration Definition (HCD), now stored in the static part of the UCW, to determine which channels and control units can reach the target device. Initial selection may be delayed if:

- All channel paths (serving the device) are busy (CP busy). The I/O request is queued at initiative queue again (in the same or from other SAP).
- ESCON Director port is busy (DP busy). SAP tries either another path (if this other path goes through another port), or the I/O request is queued in an LCU queue.
- The control unit interface, called the host adapters busy (CU busy). SAP tries either another path (if this other path goes through another logical control unit), or the I/O request is queued in an LCU queue.
The device is busy, due to shared DASD (several z/OS systems connected to the same device) activity generated in another z/OS (DV busy). The I/O request is delayed.

During all of these delays, the I/O request is serviced by the System Assist Processor (SAP) without z/OS awareness. If the channel path is not fully available, as we mentioned, SAP queues the I/O request (represented by the UCW) in some queues (depending on the type of delay).

The same I/O request may be queued several times by the same or other delay reasons. These queue are ordered by the I/O priority, as determined by the z/OS component Workload Manager (WLM).

When the I/O operation finishes (device-end status is presented), SAP queues the UCW (containing all the I/O operation final status) in the I/O interrupt queue, ready to be picked up by any enabled CPU. All the time between the SSCH and the real start of the I/O operation is timed by SAP and is called *pending time*. 
1.53 Channel processing

CCWs
The channel command word (CCW) is required to perform I/O requests to any type of device. The CCW, or chain of CCWs, is constructed by the requestor, which may be an access method or an I/O request by a program using standard assembler macros.

There may be more than 1 CCW in an I/O request to read or write data to devices, and the request is sometimes referred to as the channel program. In a channel program, (CCW) is the address of the CCW (shown as @CCW in Figure 1-53), which describes the following:

- How to locate the record in the device (cylinder, track, block)
- Physical record length (COUNT)
- Address in the memory from or to the data is moved (@DATA)
- Flags telling when the channel program ends

I/O request from a program
Programs executing in the CP initiate I/O operations with the instruction START SUBCHANNEL (SSCH).

I/O request to the subchannel
This SSCH instruction passes the contents of an operation-request block (ORB) to the subchannel. The contents of the ORB include the subchannel key, the address of the first CCW to be executed, and a specification of the format of the CCWs. The CCW specifies the command to be executed and the storage area, if any, to be used.
When the ORB contents have been passed to the subchannel, the execution of START SUBCHANNEL is complete. The results of the execution of the instruction are indicated by the condition code set in the program status word. When facilities become available, the channel subsystem fetches the first CCW and decodes it according to the format bit specified in the ORB.

Channel path selection to device
If the first CCW passes certain validity tests and does not have the suspend flag specified as a one, the channel subsystem attempts device selection by choosing a channel path from the group of channel paths that are available for selection. A control unit that recognizes the device identifier connects itself logically to the channel path and responds to its selection.

The channel subsystem sends the command code part of the CCW over the channel path, and the device responds with a status byte indicating whether the command can be executed. The control unit may logically disconnect from the channel path at this time, or it may remain connected to initiate data transfer.

Communication paths
This communication is managed by a protocol such as ESCON, FICON, FICON EXPRESS and FCP (FCP is not supported by z/OS, but is supported in zSeries for Linux), which determines the channel type and the host adapter type in the I/O control unit. Refer to “Channel subsystem (CSS) elements” on page 76 for all the channel types.

However, independently of the channel type and the I/O control unit host adapter type, the information needed by the channel to execute the I/O operation (together with the I/O control unit) is described in a channel program. Making an analogy, the CPU execute programs made of instructions, and the channel executes channel programs made of Channel Command Words (CCWs). The channel program is written by z/OS components such as VSAM, QSAM, BSAM, and BPAM.

Note: During the execution of the I/O operation, there are moments when the channel is connected to the control unit, transferring data and control information. There are other times when only the control unit is working (such as in a cache miss); this time is called disconnected.
1.54 I/O interrupt processing

Device end condition (DE)
A device end status condition generated by the I/O device and presented following the conclusion of the last I/O operation of a start function is reset at the subchannel by the channel subsystem without generating an I/O-interrupt condition or I/O-interrupt request, if the subchannel is currently start pending and if the status contains device end either alone or accompanied by control unit end (CE). If any other status bits accompany the device end status bit, then the channel subsystem generates an I/O interrupt request with deferred condition code 1 indicated.

I/O interrupts
The traditional I/O for z/Architecture is an interrupt-driven architecture (the channel interrupts the CP when the I/O completes), as follows:
- When an interrupt condition is recognized by the channel subsystem, it is indicated at the appropriate subchannel.
- The subchannel then has a condition of status pending.
- The subchannel becoming status pending causes the channel subsystem to generate an I/O-interrupt request.
- An I/O-interrupt request can be brought to the attention of the program only once. An I/O-interrupt request remains pending until it is accepted as follows:
  - By a CP in the configuration.
– It is withdrawn by the channel subsystem.
– It is cleared by means of the execution of TEST PENDING INTERRUPTION (TPI), TEST SUBCHANNEL (TSCH), or CLEAR SUBCHANNEL, or by means of subsystem reset.

When a CP accepts an interrupt request (the CP can be disabled individually by a channel through control register (CR) 6) and stores the associated interrupt code, the interrupt request is cleared and the I/O new PSW is loaded in the current PSW.

Alternatively, an I/O interrupt request can be cleared by means of the execution of TEST PENDING INTERRUPTION (TPI).

In either case, the subchannel remains status pending until the associated interrupt condition is cleared when TEST SUBCHANNEL or CLEAR SUBCHANNEL is executed, or when the subchannel is reset.

By means of mask bits in the current PSW, a CP may be enabled or disabled for all external, I/O, and machine check interruptions and for some program check interrupts. When a mask bit is one, the CP is enabled for the corresponding type of interrupt, and those interrupts can occur. When a mask bit is zero, a CP is disabled for the corresponding interrupt.

**Interruption-response block (IRB)**
The IRB is the operand of the TEST SUBCHANNEL (TSCH) instruction. The two rightmost bits of the IRB address are zeros, designating the IRB on a word boundary. The IRB contains three major fields:

- Subchannel-status word
- Extended-status word
- Extended-control word

Usually, IOS then schedules an SRB to post the task waiting IOS. This is done asynchronously for the I/O operation, and changes its state from wait to ready. Another SSCH may be executed for a previously queued I/O request.

In certain error situations, the I/O interrupt is not generated within an expected time frame. The MVS component Missing Interrupt Handler (MIH), a timer-driven routine, alerts IOS about this condition.
1.55 I/O summary

Figure 1-55 shows a summary of the flow of an I/O operation from the request issued by an application until the operation completes.

1. The user program grants access to the data set by issuing an OPEN macro, which invokes the Open routine through an SVC instruction. Later, to request either input or output of data, it uses an I/O macro like GET, PUT, READ, or WRITE, and specifies a target I/O device. These macros generate a branch instruction, invoking an access method. An access method has the following functions:
   - Writes the channel program (with virtual addresses)
   - Implements buffering
   - Guarantees synchronization
   - Executes I/O recovery

The user program could bypass the access method, but it would then need to consider many details of the I/O operation, such as the physical characteristics of the device.

2. There are several z/OS access methods, such as VSAM, BSAM, and BPAM, each of which offers differing functions to the user program. The selection of an access method depends on how the program plans to access the data (randomly or sequentially, for example).
3. To request the movement of data, the access method presents information about the operation to an I/O driver routine (usually the EXCP driver) by issuing the EXCP macro, which expands into an SVC 0 instruction.

   The I/O driver translates the channel program from virtual to real (a format acceptable by the channel subsystem), fixes the pages containing the CCWs and the data buffers, guarantees the volume extents, and invokes the I/O Supervisor (IOS).

4. IOS, if there is no pending I/O operation for the required device (originated previously in this system), issues the Start Subchannel (SSCH) instruction to the channel subsystem. Then the CPU continues with other work (the task executing the access method is probably placed in wait state) until the channel subsystem indicates with an I/O interrupt that the I/O operation has completed. If the device is busy, the request is queued in the UCB control block.

5. The SAP selects a channel path to initiate the I/O operation. This channel executes the channel program controlling the movement of data between device, control unit, and central storage.

6. When the I/O operation is complete, SAP signals the completion by generating an I/O interrupt towards any I/O-enabled CPU.

7. IOS processes the interrupt by determining the status of the I/O operation (successful or otherwise) from the channel subsystem. IOS reads the IRB to memory by issuing the TSCH instruction. IOS indicates that I/O is complete by posting the waiting task and calling the dispatcher.

8. When appropriate, the dispatcher dispatches the task returning to the access method code.

9. The access method returns control to the user program, which can then continue its processing.
Chapter 2. Introducing the IBM z10

Some of the key requirements in the commercial environment today are the need to maximize return on investments (ROI) by deploying resources designed to drive efficiencies and economies of scale; managing growth through resources that can scale to meet changing business demands; reducing risk by reducing the threat of lost productivity because of downtime or security breaches; reduce complexity by reversing the trend of server proliferation; and enabling business innovation by deploying resources that can help protect existing investments, while also enabling new technologies that enable business transformation.

The IBM System z10 Enterprise Class (z10 EC) delivers a world-class enterprise server designed to meet these business needs. The z10 EC provides new levels of performance and capacity for growth and large scale consolidation, improved security, resiliency and availability to reduce risk, and introduces just-in-time resource deployment to help respond to changing business requirements.

As environmental concerns raise the focus on energy consumption, the z10 EC is designed to reduce energy usage and save floor space when used to consolidate x86 servers. Specialty engines continue to help users expand the use of the mainframe for a broad set of applications, while helping to lower the cost of ownership. The z10 EC is at the core of the enhanced System z platform that delivers technologies that businesses need today along with a foundation to drive future business growth.

The performance design of the z/Architecture can enable the server to support a new standard of performance for applications through expanding upon a balanced system approach. As CMOS technology has been enhanced to support not only additional processing power, but also more PUs, the entire server is modified to support the increase in processing power.

The z10 EC technology shares some concepts and functions (such as HDFU) with the POWER® used in IBM p servers (RISC). The I/O subsystem supports a greater amount of bandwidth than previous generations through internal changes, providing for larger and faster volume of data movement into and out of the server. Support of larger amounts of data within the server required improved management of storage configurations, made available through integration of the operating system and hardware support of 64-bit addressing. The combined
balanced system design allows for increases in performance across a broad spectrum of work.


2.1 z196 and z10 overview

Figure 2-1   The IBM Systems196 and z10

zEnterprise highlights
zEnterprise consists of three major components:

- zEnterprise 196 central processor complex (CPC), also named z196
- zEnterprise BladeCenter Extension (also named zBX) Model 002
- zEnterprise Unified Resource Manager LIC.

zEnterprise 196 CPC (z196)
The zEnterprise 196 CPC has a newly designed quad-core chip, the fastest in the industry at 5.2 GHz. The z196 can be configured with up to 80 processors running concurrent production tasks with up to 3 TB of memory. It offers hot pluggable I/O drawers that complement the I/O cages, and continues the utilization of advanced technologies such as InfiniBand.

The z196 goes beyond previous designs while continuing to enhance the traditional mainframe qualities, delivering unprecedented performance and capacity growth. The z196 is a well-balanced general-purpose server that is equally at ease with compute-intensive and I/O-intensive workloads.

zEnterprise BladeCenter Extension
The integration of heterogeneous platforms is based on IBM's BladeCenter® technology. The IBM zEnterprise BladeCenter Extension (zBX) Model 002 houses general purpose blades as well as blades for specialized solutions.

Chapter 2. Introducing the IBM z10   107
**z10 EC**

The z10 EC, which is displayed in Figure 2-1, is designed to provide up to 1.7 times the total system capacity of the previous z9 EC, and has up to triple the available central storage (from 512 GB to 1.5 GB). The maximum number of processor units (PUs) has grown from 64 to 77. It also offers a 14% improvement in performance per KWh over the IBM System z9 EC.

All z10 EC (machine type designation 2097) models ship with two frames (twin-frame system):

- The A-Frame has two cages: a PU cage with the PUs and central storage (above) and one I/O cage (below).
- The Z-Frame has two I/O cages containing two ThinkPads used as Support Elements.

The I/O cages contain I/O cards with I/O processors, known as channels.

The processor unit (PU) cage can have up to four components, known as books. Each book is comprised of processor units (PUs), memory, and I/O fanout cards. The fanout cards are paths from the PU cage to I/O channels in the I/O cages.

The z10 EC has five configuration model offerings, from one to 64 characterized processor units (PUs); refer to “Processor unit (PU) instances” on page 113 to learn about characterized PUs. Four z10 EC models (E12, E26, E40 and E56) have 17 PUs per book, and the high capacity E64 has one 17 PU book and three 20 PU books. The configuration models vary with the number of books and the number of PUs per book.
2.2 IBM System z nomenclature

Figure 2-2  IBM System z nomenclature

IBM System z nomenclature

Figure 2-2 shows the elements of the nomenclature used to identify IBM System z servers prior to the introduction of the z196. The z196 provides 60 percent more capacity with the same energy footprint as a z10 EC, plus options for water cooling and high voltage DC power for greater energy efficiency. At the core of the zEnterprise System is the next generation mainframe—the zEnterprise 196 (z196), the industry’s fastest and most scalable enterprise system. To support a workload optimized system, the z196 can scale up (over 52,000 MIPS in a single footprint), scale out (80 configurable cores) and scale within (specialty engines, cryptographic processors, hypervisors) executing in an environmentally friendly footprint. And the z196 is designed to work together with system software, middleware and storage to be the most robust and cost effective data server. The z196 offers an industry standard PCIe I/O drawer for FICON and OSA-Express multimode and single mode fiber optic environments for increased capacity, infrastructure bandwidth, and reliability.

The z196 offers a total of 96 cores running at 5.2 GHz, and delivering up to 40 percent improvement in performance per core and up to 60 percent increase in total capacity for z/OS, z/VM, and Linux on System z workloads compared to its predecessor, the z10 EC. The z196 has up to 80 configurable cores for client use. The cores can be configured as general purpose processors (CPs), Integrated Facilities for Linux (IFLs), System z Application Assist Processors (zAAPs), System z Integrated Information Processors (zIIPs), additional System Assist Processors (SAPs), Internal Coupling Facilities (ICFs), or used as additional spares.
### 2.3 z10 EC naming summary

**z10 EC name summary**

Figure 2-3 shows the official identification of the z10 EC servers.

**System design numeric comparison**

Figure 2-3 shows a four-axis comparison between the four different product lines: z900, z990, z9 and z10. As you can see, the growth between z9 and z10 is huge but harmonious in several areas:

- MIPS per PU grew 1.62 times. Using Large Systems Performance Reference (LSPR) measurements, new measurements with z/OS V1R8, and reevaluating the calculation for the mixed workload, the z10 EC uni-processor MIPS is 920.
- The maximum number of processors grew from 54 to 64 and PUs from 64 to 77.
- Central storage grew from 512 GB to 1.5 TB.
- The aggregate I/O data rate grew from 172.8 to 288 GB/sec.

Central storage growth is proportionally larger than the other three resources. The justification is that in a heavy commercial data processing environment, central storage can really improve the overall performance by drastically decreasing the I/O rate. In other words, central storage is good for the performance health of the full system.
2.4 The power of GHz (high frequency)

Figure 2-4   The power of the GHz

Does GHz (high frequency) matter

High GHz (or low cycle time) is not the only dimension that matters when talking about processor performance; system performance is not linear with frequency. Instead, you need to use Large Systems Performance Reference (LSPR) measurements along with System z capacity planning tools to achieve real client and workload sizing.

The processor speed for executing a program depends on:

- Low cycle time and consequently high GHz (frequency) - as provided by the technology (type of circuitry).
- Low number of cycles per average executed instruction (CPAI) - as provided by the design (pipeline, cache).
- Fewer instructions in the referred program - as provided by the architecture (powerful set of instructions).

The z10 EC focus is on balanced system design and architecture to leverage technology and exploit high frequency design, as listed here:

- Low-latency pipeline (refer to “Pipeline in z10 EC” on page 129 for more information).
- Dense packaging (MCM) allows MRU cooling, which yields more power-efficient operation.
- Optimum set of instructions.
Virtualization technology allows consistent performance at high utilization, which makes processor power-efficiency a much smaller part of the system and data center power consumption picture.

Environmentally sensitive energy efficiency.

However, frequency (GHz) is still significant and important because it is a way for hardware to speed up the processors.

System z has steadily and consistently grown frequency, with occasional jumps/step functions (G4 in 1997, z10 in 2008).
2.5 Processor unit (PU) instances

- Central processor (CP)
- Integrated facility for Linux (IFL)
- Integrated Coupling Facility (ICF)
- z10 Application Assist Processor (zAAP)
- z10 Integrated Information Processor (zIIP)
- System Assisted Processor (SAP)
- Spare

Processor unit (PU) instances

All PUs are physically identical. However, at Power-on Reset, it is possible to have different types of PUs through a small LIC change (for the SAP PU, the change is large), leading then to different instances. These instances are also called specialty engines. The major reason for such engines is to lower overall total cost of computing in the mainframe platform. They can be ordered by customers and are known as characterized. Following are the possible PU instances:

- **CPU**: A processor is a general purpose processor that is able to execute all the possible z10 EC running operating systems, as such z/OS, Linux, z/VM, z/VSE, Coupling Facility Control Code (CFCC), and z/TPF. A CPU is also known as a CP.
- **IFL**: This type of PU is only able to execute native Linux and Linux under z/VM. IFLs are less expensive than processors.
- **ICF**: This type of PU is only able to execute CFCC operating system. The CFCC is loaded in a Coupling Facility LP from a copy in HSA; after this, the LP is activated and IPLed. ICFs are less expensive than processors.
- **zAAP**: This type of PU only runs under z/OS, and is for the exclusive use of Java interpreter code (JVM) and DB2 9 XML parsing workloads. A zAAP is less expensive than a processor, and does not increase the software price (based on produced MSUs) because it does not produce MSUs.
- **zIIP**: This type of PU is run in z/OS only, for eligible DB2 workloads such as DDF, business intelligence (BI), ERP, CRM and IPSec (an open networking function...
used to create highly secure crypto connections between two points in an enterprise) workloads. A zIIP is less expensive than a processor and does not increase the software price (based on produced MSUs) because it does not produces MSUs. There is a limitation about using zIIP processors, that is, only a percentage of the candidate workload can be executed.

**SAP**
A System Assist Processor (SAP) is a PU that runs the Channel Subsystem Licensed Internal Code. An SAP manages the starting of I/O operations required by operating systems running in all logical partitions. It frees z/OS (and the processor) from this role, and is “mainframe-unique”. z10 EC models have a variable number of standard SAPs configured.

**Spare**
A spare PU is a PU that is able to replace, automatically and transparently, any falling PU in the same book, or in a different book. There are at least two spares per z10 EC server.

**Ordering z10 EC models**
When a z10 EC order is configured, PUs are characterized according to their intended usage. They can be ordered also for future growth, as follows:

**Capacity marked CP**
A CP that is purchased for future use as a CP is marked as available capacity. It is offline and unavailable for use. A capacity marker identifies that a certain number of CPs have been purchased. This number of purchased CPs is higher than the number of CPs actively used. The capacity marker marks the availability of purchased but unused capacity intended to be used as CPs in the future; they usually have this status for software charging reasons. Unused CPs do not count in establishing the MSU value to be used for WLC plan for software charging, or when charged on a per server basis.

**Unassigned IFL**
A PU purchased for future use as an IFL, offline, unavailable for use.

**Unassigned ICF**
A PU purchased for future use as an ICF, offline, unavailable for use.

**Unassigned zAAP**
A PU purchased for future use as a zAAP, offline, unavailable for use.

**Unassigned zIIP**
A PU purchased for future use as a zIIP, offline, unavailable for use.

**Additional SAP**
The optional System Assist Processor is a PU that is purchased and activated for use as an SAP if your I/O workload demands it.

The development of a multibook system provides an opportunity to concurrently increase the capacity of the system in several areas:

- You can add capacity by concurrently activating more characterized PUs, such as CPs, IFLs, ICFs, zIIPs, zAAPs or SAPs, on an existing book.
- You can add a new book concurrently (non-disruptively) and activate more CPs, IFLs, ICFs, zIIPs, zAAPs or SAPs.
- You can add a new book to provide additional memory and fanout cards to support increasing storage and/or I/O requirements.

**Simulation support**
z/VM guest virtual machines can create virtual specialty processors on processor models that support same types of specialty processors but do not necessarily have them installed. Virtual specialty processors are dispatched on real CPs. Simulating specialty processors provides a test platform for z/VM guests to exploit mixed-processor configurations. This allows users to assess the operational and processor utilization implications of configuring a
z/OS system with zIIP or zAAP processors without requiring the real specialty processor hardware.
2.6 z10 EC hardware model

The z10 EC server hardware model nomenclature (also called Configs) is based on the number of PUs available for customer use in each server. These PUS are called characterized and can take any PU personality. The z10 EC has five models with a total of 100 capacity settings available:

Model E12: one book with 17 PUs, 3 standard SAPs and 2 standard spare, then 12 characterized PUs.

Model E26: two books with 17 PUs each, 6 standard SAPs and 2 standard spare, then 26 characterized PUs.

Model E40: three books with 17 PUs each, 9 standard SAPs and 2 standard spare, then 40 characterized PUs.

Model E56: four books with 17 PUs each, 10 standard SAPs and 2 standard spare, then 56 characterized PUs.

Model E64: one book with 17 and three books with 20 PUs each, 11 standard SAPs and 2 standard spare, then 64 characterized PUs.

As with z9 servers, the z10 EC hardware model names indicate the maximum number of processor units potentially orderable (named characterized), and not the actual number of active processors.

### Processor Unit Features

<table>
<thead>
<tr>
<th>Model</th>
<th>Books/ PUs</th>
<th>CPs</th>
<th>IFLs/ uIFLs</th>
<th>zAAPs/ zIIPs</th>
<th>ICFs</th>
<th>Standard SAPs</th>
<th>Standard Spares</th>
</tr>
</thead>
<tbody>
<tr>
<td>E12</td>
<td>1/17</td>
<td>0-12</td>
<td>0-12 0-11</td>
<td>0-6 0-6</td>
<td>0-12</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>E26</td>
<td>2/34</td>
<td>0-26</td>
<td>0-26 0-25</td>
<td>0-13 0-13</td>
<td>0-16</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>E40</td>
<td>3/51</td>
<td>0-40</td>
<td>0-40 0-39</td>
<td>0-20 0-20</td>
<td>0-16</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>E56</td>
<td>4/68</td>
<td>0-56</td>
<td>0-56 0-55</td>
<td>0-28 0-28</td>
<td>0-16</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>E64</td>
<td>4/77</td>
<td>0-64</td>
<td>0-64 0-63</td>
<td>0-32 0-32</td>
<td>0-16</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

A minimum of one CP, ICF, or ICF must be purchased on every model. One zAAP and one zIIP may be purchased for each CP purchased.

Figure 2-6  z10 EC hardware model
2.7 z10 EC sub-capacity models

Sub-capacity models
Sub-capacity models (also called sub-unis) are servers in which the processor speed is artificially downgraded due to financial and marketing considerations. In z10 EC there are a total of 36 sub-capacity settings. They are only available for the E12 models (from 1 to 12 processors). Models with 13 CPs or greater must be full capacity.

For the E12 models, there are four capacity settings per processor. The entry point (Model 401) is approximately 23.69% of a full speed processor (Model 701). All specialty engines (zAAP, zIIP, ICF, IFL) continue to run at full speed. Sub-capacity processors have all the availability of z10 EC features and functions. All processors must be the same capacity setting size within one z10 EC.

Software model numbers
For software billing purposes only, there is a Capacity Indicator (also called a software model number) associated with the number of PUs that are characterized as processors. This number is stored in memory by the Store System Information (STSI) instruction.

There is no affinity between the hardware model and the number of activated CPs. For example, it is possible to have a Model E26 (26 characterized PUs) but with only 13 processors, so for software billing purposes, the STSI instruction would report 713.
The software model number (Nxx) follows these rules:

- **N** = the capacity setting of the engine:
  - 7xx = 100%
  - 6xx ~ 69.35%
  - 5xx ~ 51.20%
  - 4xx ~ 23.69%

- **xx** = the number of PUs characterized as processors in the server

After **xx** exceeds 12, then all processor engines are full capacity. Here is a list for all possible values:

- 700, 401 to 412, 501 to 512, 601 to 612 and 701 to 764.

  The software model 700 does not have any processor engines, it may have ICFs and IFLs.

There are 101 \((64 + 36 +1)\) possible combinations of capacity levels and numbers of processors. These offer considerable overlap in absolute capacity, provided different ways. For example, a specific capacity (expressed as MSUs) might be obtained with a single faster CP or with three slower CPs. The hardware cost is approximately the same. The single-CP version might be a better choice for traditional CICS® workloads because they are single task (however, a processor loop can have a significant negative impact), and the three-way server might be a better choice for mixed batch workloads.
2.8 z10 EC frames and cages

As previously mentioned, all z10 EC models ship with two frames:

- The A-Frame is divided into two cages: a PU cage (above) with the processor units (up to 77) and central storage (up to 1.5 TB), and one I/O cage (below), plus power and cooling units (MRU) and an optional internal battery feature (IBF).
- The Z-Frame is divided into two optional I/O cages containing two ThinkPads that are used as Support Elements (a processor controlling the server and used as an interface with IBM custom engineers), plus basic power supplies and an optional IBF.

The I/O cages have I/O cards (located in I/O slots) containing I/O processors known as channels. In total you may have up to 1024 channels, but up to 256 accessed per z/OS image.

The processor unit (PU) cage (also called the CEC cage) can have up to four components known as books connected via a point-to-point SMP network. (These components are called books because they look like books on a library shelf.) Each book is made of up to 20 processor units (PUs), memory (up to 384 GB) and I/O fanout cards. The fanout cards are paths to I/O channels in the I/O cages.

z10 models

The z10 EC has five configuration model offerings, from one to 64 characterized processor units (PUs). Four z10 EC models (E12, E26, E40 and E56) have 17 PUs per book, and the
high capacity E64 has one 17 PU book and three 20 PU books. We have up to standard 11 SAPs and always two Spare PUs per server.

The configuration models vary with the number of books and the number of PUs per book.

**Optional battery feature (IBF)**

IBF keeps the server powered up for up to 10 minutes when there is a power outage (in all four AC feeds). This time amount is dependent on the number of I/O cages. In practical terms, the IBF can be used as follows:

- To keep the storage contents of the LP running the non-volatile Coupling Facility (CF), then allowing structures rebuild in the other CF.
- For orderly shutdown of z/OS in case of a longer outage, if the I/O storage configuration has an alternate source of power (which is usually the case).
- To avoid disruption while waiting for a short power outage to pass.

**I/O cages**

The z10 EC contains an I/O subsystem infrastructure which uses an I/O cage that provides 28 I/O slots and the ability to have one to three I/O cages, delivering a total of 84 I/O slots.

The z10 EC continues to use the Cargo cage for its I/O (as used by the z9 servers), supporting up to 960 ESCON and 256 FICON channels on the Model E12 (64 I/O slots) and up to 1024 ESCON and 336 FICON channels (84 I/O slots) on the Models E26, E40, E56 and E64.
2.9 Book topology comparison

As shown on the right side of Figure 2-9, in the z10 EC server there are 77 PUs possible, and all books are interconnected in a star configuration with high speed communications links via the L2 caches. This allows the system to be operated and controlled by the LPAR facility as a symmetrical, memory-coherent multiprocessor. It was designed to get the maximum benefit of the improved processor clock speed.

Books are interconnected by a point-to-point connection topology. This allows every book to communicate with every other book. Data transfer never has to go through another book (cache) to address the requested data or control information. Inter-book communication takes place at the Level 2 (L2) cache level.

The L2 cache is implemented on two Storage Control (SC) cache chips in each MCM. Each SC chip holds 24 MB of SRAM cache, resulting in a 48 MB L2 cache per book. The L2 cache is shared by all PUs in the book and has a store-in buffer design. The SC function regulates coherent book-to-book traffic.

The ring topology employed on the z9 EC server (64 PUs) and shown on the left in the figure is not used on the z10 EC.

In a system with more than one book, all physical memory in the book containing the failing memory is taken offline, which allows you to bring up the system with the remaining physical memory in the other books. In this way, processing can be resumed until a replacement memory card is installed.
2.10 NUMA topology

NUMA topology
In a symmetrical multiprocessing (SMP) design, as implemented in z10 EC where several PUs (up to 77) access (share) the same central storage, there is a limitation for the numbers of such processors, performance-wise. This limitation is much smaller than 77.

To overcome such a problem, Non-Uniform Memory Access (NUMA) topology in z10 EC servers is introduced. Each PU has local access to one piece (25%) of central storage (the one allocated in its book) and remote access to other pieces (75%). In other words, the access is non-uniform.

LPAR logic and the z/OS dispatcher together try to guarantee that the majority (80%) of the memory access by one PU is in the same book.

As an analogy, consider a baker who decides spontaneously to bake a cake, but only has flour, sugar, and eggs (80% of the ingredients) at home, because it is uneconomical to keep all the ingredients on hand. So the baker needs to go (remotely) to a supermarket for the remaining 20% of the necessary ingredients.
2.11 z10 EC Books

A book has processor units (PUs), central storage, and fanout cards connecting the book with the I/O cages, where the I/O channels are. Through the fanout cards:

- A channel can access data (read or write) in central storage, moving it to or from I/O controllers along I/O operations.
- SAPs can talk to channels during the start of the I/O operation or I/O interrupts.

Books are located in the PU cage in Frame A (see Figure 2-11), and have the following components:

- Multi-chip module (MCM). Each MCM includes five quad-core processor unit (PU) chips and two Storage Control (SC) chips. Refer to “Multi-chip module (MCM)” on page 125 for more information about this topic.
- Memory dual in-line memory module (DIMM). The DIMM is plugged into 48 available slots, providing 64 GB to 384 GB of central storage.
- A combination of up to eight InfiniBand host channel adapters (HCA2-Optical or HCA2-Copper) and memory bus adapter (MBA) fanout cards can be installed. Each one has two ports, supporting up to 16 connections. HCA2-copper connections are for links to I/O cages in this server. HCA2-Optical and MBA are for links to external servers (coupling links). MBA cards are used for ICB-4 links only. Refer to “Connecting PU cage with I/O cages” on page 149 for more information about these fanout cards.
Distributed converter assemblies (DCAs). Three DCAs provide power to the book. Loss of a DCA leaves enough book power to satisfy the books’ power requirements. The DCAs can be concurrently maintained.

Functional service processor (FSP). The FSP card is based on the IBM Power PC microprocessor. It connects to an internal Ethernet LAN to communicate with the support elements in this book and in other books.

About book replacement
With enhanced book availability and flexible memory options, a single book in a multibook system can be concurrently removed and reinstalled for an upgrade or repair; it is a field replacement unit (FRU). Any book can be replaced, including book 0, which initially contains the HSA.

However, this requires that you have sufficient resources in the remaining books to avoid impacting the workload. CPs and memory from the book must be relocated before the book can be removed. Not only do additional PUs need to be available on the remaining books to replace the deactivated book, but also sufficient redundant memory must be available if it is required that no degradation of applications is allowed.

You may want to consider using the flexible memory option. Removal of a book also cuts the book connectivity through its host channel adapters (HCA); refer to “Connecting PU cage with I/O cages” on page 149 for more information about the HCA. The impact of the removal of the book is limited by the use of redundant I/O Interconnect. However, all MBAs on the removed book have to be configured offline.

PR/SM™ has knowledge of the amount of purchased memory and how it relates to the available physical memory in each of the installed books. PR/SM also has control over all physical memory and therefore is able to make physical memory available to the configuration when a book is non-disruptively added.

PR/SM also controls the reassignment of the content of a specific physical memory array in one book to a memory array in another book. This is known as the Memory Copy/Reassign function. It is used to reallocate the memory content from the memory in a book to another memory location when enhanced book availability is applied, to concurrently remove and reinstall a book in case of an upgrade or repair action.

Also, PR/SM always attempts to allocate all real storage for a logical partition within one book.
2.12 Multi-chip module (MCM)

The z10 EC book has a new Multi-Chip Module with five new IBM z10 Processor chips. Depending on the MCM version (17 PU or 20 PU), from 17 to 77 PUs are available, on one to four books. This new MCM provides a significant increase in system scalability and an additional opportunity for server consolidation. All books are interconnected with very high speed internal communications links in a fully connected star topology via the L2 cache, which allows the system to be operated and controlled by the PR/SM facility as a symmetrical, memory-coherent multiprocessor.

There is just one MCM per book in a z10 EC server. It contains seven chips and measures approximately 96 x 96 millimeters. All chips use Complementary Metal Oxide of Silicon (CMOS) 11S chip technology. CMOS 11s is state-of-the-art microprocessor technology based on ten-layer Copper Interconnections and Silicon-On Insulator technologies.

An MCM contains 103 glass ceramic layers to provide interconnection between the chips and the off-module environment. It produces cycle time of approximately 0.23 nanoseconds, that is, 4.4 GHz for frequency. Each MCM has:

- Five PU chips, and each PU chip has up to four PUs (engines).
  - Two MCM options are offered: with 17 PUs, or 20 PUs.
- Two SC chips, and each SC chip is connected to all five PU chips.
2.13 PU chip

The new Enterprise Quad Core z10 PU chip is comprised of three or four PUs. It contains 6 Km of wire, and has 994 million transistors in a 450-mm² area. A schematic representation of the PU chip is shown in Figure 2-13.

As mentioned, there are five PU chips on each MCM. The five PU chips come in two versions. For models E12, E26, E40, and E56, the processor units on the MCM in each book are implemented with a mix of three active cores and four active cores per chip (3 x 3 cores active, plus 2 x 4 cores active), resulting in 17 active cores per MCM. All MCMs in these models have 17 active cores. This means that a Model E12 has 17, a Model E26 has 34, a Model E40 has 51, and a Model E56 has 68 active PUs.

The four PUs (cores) are shown in the corners of the figure. They include the L1 and L1.5 caches, plus all microprocessor functions. COP indicates the two co-processors, each of which is shared by two of the four cores. The co-processors are accelerators both for data compression (Zivv Lampel algorithm) and cryptographic functions. If the co-processor is busy performing compression, the crypto request must wait, and vice versa. (Do not confuse this co-processor with the Crypto Express-2 that is located in the I/O cage.)

L2 cache

The L2 cache (in the SC chip in the same MCM) interface is shared by all four PUs. GX indicates the I/O bus controller that controls the interface to the Host Channel adapters.
accessing the I/O. The Memory Controller (MC) is the gate to access the central storage banks located in this book. L2 Intf is the interface to SC and the L2 cache.

As you can imagine, there is intense traffic. This chip controls traffic between the microprocessors (PUs), memory, I/O and the L2 cache located on the SC chips.

L1 and L1.5 cache
Each PU has a 192 KB on-chip Level 1 cache (L1) that is split into a 64 KB L1 cache for instructions (I-cache) and a 128 KB L1 cache for data (D-cache). A second level on chip cache, the L1.5 cache, has a size of 3 MB per PU. The two levels on chip cache structure are needed to optimize performance so that it is tuned to the high frequency properties of each of the microprocessors (cores).

MCM and spare PUs
In each MCM, 12 to 16 available PUs may be characterized for customer use. Up to three SAPs may reside in an MCM; how many are used depends on the model and the book in which they reside. System-wide, two spare PUs (cores) are available that may be allocated on any MCM in the system. Up to two spare PUs (cores) may be allocated on an MCM.
2.14 Book element interconnections

Figure 2-14 shows a logical view of the connections within the MCM book. The z10 EC book has a new Multi-Chip Module with five new IBM z10 Processor chips. This new MCM provides a significant increase in system scalability and an additional opportunity for server consolidation.

Depending on the MCM version (17 PU or 20 PU), from 17 to 77 PUs are available, on one to four books. As previously mentioned, all books are interconnected with very high speed internal communications links, in a fully connected star topology via the L2 cache, which allows the system to be operated and controlled by the PR/SM facility as a symmetrical, memory-coherent multiprocessor.

The PU configuration is made up of two spare PUs per server and a variable number of System Assist Processors (SAPs), which scale with the number of books installed in the server—three with one book installed, and up to eleven when four books are installed. The remaining PUs can be characterized as central processors (CPs), Integrated Facility for Linux (IFL) processors, System z10 Application Assist Processors (zAAPs), System z10 Integrated Information Processors (zIIPs), internal Coupling Facility processors, or additional SAPs.

GX refers to the adapters connecting with the I/O cages.

Next, we introduce the pipeline concept.
2.15 Pipeline in z10 EC

Pipeline in z10 EC
To build a car, serial tasks need to be executed. Similarly, when executing program instructions, serialized tasks also need to be executed. The term “pipeline” implies executing, in parallel, different tasks for different cars (or instructions). The main objective of a pipeline is to increase **throughput** (that is, the number of executed instructions per time unit), and not to decrease the average time to execute one instruction.

Within the z10 EC PU there are a few special processors, each one executing a very specific function:

- Get the instruction virtual address in the PSW.
- Translate this virtual address to real address through the DAT.
- Fetch the instruction from cache (L1 or L 1.5 or L2) or from central storage.
- Decode the instruction; that is, if the instruction is microcoded, find the microcode address (in control storage) associated with its execution.
- If there is an input storage operand, calculate its virtual address through the contents of the base register plus the displacement.
- Translate this virtual address to a real address through the DAT.
- Fetch the operand from cache (L1 or L1.5 or L2) or from central storage.
- Execute the instruction.
If there is an output storage operand, derive its virtual address through the contents of the base register plus the displacement.

- Translate this virtual address to a real address through the DAT.
- Store the output operand in cache (L1 and L1.5).
- Depending on the instruction, set the condition code in the PSW.

**Techniques for instruction pipeline**

Techniques exist for speeding up an instruction pipeline:

- Execute more than one instruction in the same cycle (superscalar).

  This is implemented by adding resources onto the server to achieve more parallelism by creating multiple pipelines, each working on their own set of instructions. A *superscalar* server is based on a multi-issue architecture. In such a server, where multiple instructions can be executed at each cycle, a higher level of complexity is reached because an operation in one pipeline may depend on data in another pipeline. A superscalar design therefore demands careful consideration of which instruction sequences can successfully operate in a multi-pipeline environment.

  The z10 EC PU is superscalar, which means that it is able to decode up two instructions at the same time and to execute up to three (depending on the instructions).

- Perform out-of-order execution.

  This implies that the sequence of instructions presented in a program is not the sequence where the instructions are executed. For example, if the instruction (n+1)th is ready to be executed and the nth instruction is still being delayed by a storage operand fetch, and the result of n does not interfere in the input of n+1, then n+1 is executed first. z10 EC PU does not implement such function; rather, it performs in-order execution.

- Perform out-of-order fetch.

  Instructions having memory operands may suffer multi-cycle delays to get the memory content. To overcome these delays, the server continues to fetch (single cycle) instructions that do not cause delays. The technique used is called *out-of-order operand fetching*.

  This means that some instructions in the instruction stream are already under way, while earlier instructions in the instruction stream that cause delays due to storage references take longer. Eventually, the delayed instructions catch up with the already-fetched instructions and all are executed in the designated order. The z10 EC PU implements such function.
2.16 Pipeline branch prediction

Pipeline branch prediction
z10 EC performs branch prediction through a branch history table (BHT). The branch history table implementation on processors has a large performance improvement effect, but is not sufficient for the z10 EC. The BHT was originally introduced on IBM ES/9000 9021 in 1990 and has been improved ever since. The BHT offers significant branch performance benefits. The BHT allows each PU to take instruction branches based on a stored BHT, which improves processing times for calculation routines. In addition to the BHT the z10 EC uses a variety of techniques to improve the prediction of the correct branch to be executed.

Every conditional branch instruction in a program poses a problem to a pipeline at the fetch instruction moment: which side of the branch will be taken at conditional branch execution time, and which side of the fork will fetch instructions? To address this the pipeline remembers, when the same code was executed in the past, whether that conditional branch had branched or simply went straight in the code. Such information will be used in the fetch and decode decision. The mechanisms used are:

- Branch history table (BHT)
- Branch target buffer (BTB)
- Pattern history table (PHT)
- BTB data compression

The success rate of branch prediction contributes significantly to the superscalar aspects of the z10 EC, given the fact that the architecture rules prescribe that for successful parallel execution of an instruction stream, the correctly predicted result of the branch is essential.
2.17 About each z10 EC PU

Figure 2-17 shows an actual picture of a z10 EC PU inside the PU chip contained in an MCM. The highlighted portion is the HDFU. Each PU (core) has its own decimal floating point unit (HDFU). The L2 cache interface is shared by all four cores. MC indicates the memory controller function controlling access to memory, and GX indicates the I/O bus controller that controls the interface to the host channel adapters accessing the I/O. The chip controls traffic between the microprocessors (cores), memory, I/O, and the L2 cache on the SC chips.

A PU is superscalar when it may execute more than one instruction per cycle. Then, z10 EC PU is superscalar because it may execute (finish) up to three instructions (depending on the specific instructions) per cycle (230 picoseconds).

The PU is able to execute 894 different instructions, as described in z/Architecture Principles of Operations, SA22-7832. More than 50+ instructions are added to z10 EC to improve object code efficiency, such as for Java BigDecimal, C#, XML, XL C/C++, GCC, DB2 V9, and Enterprise PL/1. To achieve high speed, 668 of such instructions are implemented entirely in hardware, and only 226 instructions are microcoded and millicoded.

Millicode is a PU internal code at higher level than microcode and is very similar to Assembler. Millicode is required to implement the more complex instructions of the instruction set, such as Start Interpretive Execution (SIE).
HDFU
Each PU has a Hardware Decimal Floating Point unit (HDFU) to accelerate decimal floating point transactions. This function does not exist in z9 EC. It conforms with the standard IEEE 745R. This is expected to be particularly useful for the calculations involved in many financial transactions (previously they were performed by software routines).

Out of the PUs but in the PU chip, there are two co-processor units. Each co-processor unit implements cryptographic and data compression functions, and each unit is shared by two of the four PUs.

Each PU contains:

- Store though L1 cache (refer to “Three levels of cache” on page 137) divided into a 64 KB cache for instructions and a 128 KB cache for data.
- Store through L1.5 cache of 3 MB.
  
  The two levels of chip cache structure are needed to optimize performance so that it is tuned to the high frequency properties of each PU.
- Translation Look-aside Buffer (TLB) of 512 entries. These entries are used by dynamic address translation (DAT) to keep the real addresses of the 512 most-referenced pages.
- Several other processors in order to implement a pipeline (for example, an I-unit for fetching and decoding; an E-unit for execution; a unit containing L1 and L1.5 caches, and others).
2.18 z10 EC storage controller (SC) chip

The z10 EC is built on a proven superscalar microprocessor architecture. On each book there is one MCM that supports up to 20 PUs. The MCM has five PU chips and two SC chips. Each PU chip has up to four cores, which can be characterized as CPs, IFLs, ICFs, zIIPs, zAAPs, or SAPs. Two MCM sizes are offered, 17 or 20 cores, on five PU chips.

There are two SC chips on the MCM. Each SC chip contains 1.6 billion transistors. An SC chip has:

- 24 MB of L2 cache, resulting in a combined L2 cache size of 48 MB (2 x 24) per book.
- A Storage Controller processor, which controls:
  - The communication between the L1.5 cache in the PUs and the L2 cache in the same MCM is done by five bidirectional 16-byte data buses.
  - The communication of the cross-point switch for L2-to-L2 traffic to up to three remote books by three bidirectional 16-byte data buses.

Storage Controller function is described in more detail in the following section.

**Storage Controller function**

The Storage Controller (SC) acts as a coherency manager. It is responsible for coherent traffic between the L2 caches in a multi-book system, and between the L2 cache and the local PU L1 and L1.5. Note that L2 is shared between all PUs of a book.
The Storage Controller has directory information for all L1 and L1.5 caches in all the book PUs. If there is a change in the contents, the Storage Controller can invalidate the old copy in other L1 caches. It also optimizes cache traffic and does not look for cache hits in other books when it knows that all resources of a given logical partition are available in the same book. The SC chip also controls the access and storage of data in between the central storage (L3) and the L2 on-chip cache.

The SC chip also acts as an L2 cache cross-point switch for L2-to-L2 traffic to up to three remote MCMs or books by three bidirectional 16-byte data buses with a 3:1 bus/clock ratio. The SC chip measures 21.11 x 21.71 mm and has 1.6 billion transistors. The L2 SRAM cache size on the SC chip measures 24 MB, resulting in a combined L2 cache size of 48 (2 x 24) MB per book. The clock function is distributed between both SC chips, and the wire length of the chip amounts to 3 km.

Figure 2-18 on page 134 displays the various elements of the PU chip. In the figure, PIPEx is the L2 cache, and L2C is the Storage Controller itself. Most of the space is taken by the SRAM L2 cache. L2C indicates the controller function of the chip for point-to-point interbook communication. Directory and addressing function locations are also shown.
2.19 Recapping the z10 EC design

Figure 2-19 displays the major z10 EC components (there are also two frames, which are not pictured here).

The book, which is a component of the PU cage, is shown in the top right corner.

There are four books. One MCM is shown, with five PU chips and two SC chips.

Each PU chip has up to four PUs, each with L1 and L1.5 caches and HDFU, as well as two co-processors and crypto hardware functions.

Each SC chip has L2 cache and an SC processor.
Three levels of cache

Caches are fast memories used to avoid the need to access lower-level memories that are cheaper, larger and slower. In a PU, the cache is used to decrease access to relatively slow central storage. In the z10 EC, there are three levels of cache (L1, L1.5, and L2) to improve performance by reducing access to real storage (called L3). Caches are not directly visible to programs.

Cache sizes are being limited by ever-decreasing cycle times because they must respond quickly without creating bottlenecks. Access to large and distant caches cost more short cycles. This phenomenon of shrinking cache sizes can be seen in the design of the z10 EC, where the L1 instruction and data caches have been shortened to accommodate decreased cycle times.

The distance to remote caches is also a consideration; for example, the L2 cache, not located in the PU, might be in another book. To address this in the z10 EC, the L1.5 cache has been introduced. This intermediate-level cache reduces traffic to and from the L2 cache. With the L1.5 cache, requests are sent to the L2 cache only when there is a cache miss in L1 and L1.5.

Each PU has its own 192 KB Cache Level 1 (L1), split into 128 KB for data (D-cache) and 64 KB for instructions (I-cache). The reason for splitting data and instructions is because the pattern and nature of the reference is different; for example, data is often changed, but instructions are very seldom changed. The PU only fetches and store instructions or
operands (data) from or to the L1 cache. L1 cache is designed as a store-through cache, meaning that altered data is immediately (synchronously) stored to the next level of memory.

The next level of memory is the L1.5 cache, which is in each PU and is 3 MB in size. It is also a store-through cache to L2.

The MCM also contains L2 cache located in two Storage Control (SC) chips. Each SC chip has a Level 2 (L2) cache of 24 MB, for a total of 48 MB. Each L2 cache has a direct path to each of the other L2 caches in remote books on one side, and each of the PUs in the MCM on the other side, through point-to-point (any-to-any) connections.

The L2 cache is shared by all PUs within a book. SC processor services provide the communication between L2 caches across books. The L2 cache has a store-in buffer design, which means that the changes are not immediately copied to central storage (L3). The destage to central storage is performed when L2 occupancy reaches some threshold occupancy. The performance advantage of being store-in is that the PU does not need to wait for a slow store in memory. Also, if the same data is updated several times when in an L2 cache, the store to central storage performed done just once.

The z10 EC uses a least-recently used (LRU) algorithm to expel the least referenced set of operands and instructions from the L1 cache to the L1.5 cache. The same LRU is also used from L1.5 to L2, and from L2 to central storage.

Whenever an operand or instruction needs to be fetched, the L1 cache is inspected first. When the operand or instruction is not found, it is known as a “miss”. In the rare case of a miss (less than 10%), the search sequence is L1.5, L2, and then central storage.

**Changing the contents of the L1 cache**

When a PU executes an instruction that changes the contents of the output operand in its L1 cache, the following actions are executed:

- Posting the other PUs, which may have in their L1 or L1.5 cache an out-of-date copy of the storage element changed by the first PU.

  In the z10 EC, the SC located in each MCM book has the L1 cache directories of all PUs of its book. When a PU alters the contents of one of its L1 cache elements, the SC is directed to invalidate this element in the L1 cache directory of other PUs in the same book, but without disturbing those PUs. This update information is passed to the other SCs (in other books) to execute the same invalidation, if needed.

  However, just switching on the bit does not guarantee that all PUs will see the most recently updated copy at the same time. To guarantee the full coherency of memory, the program must execute serializing instructions, as described in *z/Architecture Principles of Operations*, SA22-7832.

  With the execution of such instructions (such as the Compare and Swap instruction), all previous storage access is completed, as observed by other PUs and channels, before the subsequent storage accesses occur.

- Defining a place where the other PUs can fetch the most updated copy of such contents, if they need to do so. In a z10 EC, the updated contents are copied synchronously (store-through) from the L1 cache to the L1.5 cache, and from the L1.5 cache to the L2.
cache, which is a book’s “global memory” where other PUs can reach the most updated copy of the element.

**Cache differences**

In addition to location, there are other differences between the L1/L1.5 cache and the L2 cache which make L1/L1.5 even faster than L2.

For the L1/L1.5 cache:

- There is error detection through parity bit.
- There is no local error correction (data is recovered from L2).

For the L2 cache:

- There is error detection through ECC.
- There is local error correction (which makes it slower).
2.21 Software/hardware cache optimization

Figure 2-21 HiperDispatch example

Software and hardware cache optimization

To achieve acceptable PU performance, the z10 EC design has the following rules:

- A very high percentage of instructions and data must be fetched from the L1/L1.5 caches (hits). A target of more than 90% is desirable.
- In the case of an L1/L1.5 miss, it is expected that more than 90% of such (previous) misses will be honored from the local (same book) L2 cache.
- In the case of a local L2 miss, it is expected that more than 90% of such (previous) misses will be honored from remote (other book) L2 caches.
- In the case of a remote L2 miss, it is expected that more than 90% of such (previous) misses will be honored from local (same book) central storage.

As you can see, only in a very small percentage of fetches may the instruction or operand be fetched from the central storage of other books (the worst performance case).

To achieve such design goals, several functions are implemented:

- Each SC tries to keep the data it fetched (based on this book’s PU requests) from the central storage or L2 of another book in its L2 cache book. In this way, the next fetch request to the same data will be L2 local. (Note that the same piece of data has only one copy in all four L2 caches, to avoid using serialization mechanisms to guarantee integrity.)
- LPAR always attempts to allocate all central storage for a logical partition within one book, and attempts to dispatch a logical PU from this logical partition on a physical PU from that
Each LPAR tries to dispatch the same logical processor in the same physical processor.

- The z/OS dispatcher will try to dispatch the same TCB/SRB in the same logical processor, or at least in the same group of logical processors.

HiperDispatch

The last three functions are called HiperDispatch. HiperDispatch assures that, as much as possible, the same TCB/SRB is dispatched in the same logical processor and the same logical processor is dispatched in the same physical processor. This smart dispatching tries to decrease the number of addresses referenced by a physical processor. This will improve directly the use of cache, and consequently the use of TLBs and ALBs.

On the z10 EC with HiperDispatch, this is taken one step further: PR/SM and z/OS now work in tandem to more efficiently use processor resources.

HiperDispatch is available only with the new z10 EC PR/SM through the vertical processor management (VCM) component and specialized z/OS V1R9 functions.

This function, as exploited by the z/OS dispatcher, is described in z/Architecture Principles of Operations, SA22-7832, under the name Configuration Topology Facility. This facility provides additional topology awareness to the operating system so that certain optimizations can be performed to improve cache hit ratios and thereby improve overall performance.

HiperDispatch combines the dispatcher actions and the knowledge that PR/SM has about the topology of the PUs in the server. For that purpose, the z/OS dispatcher manages multiple TCB/SRB queues with an average number of four logical processors per queue. It uses these queues to assign work to as few logical processors as are needed for a given LPAR workload. So even if the LPAR is defined with a large number of logical processors, HiperDispatch will optimize this number of processors nearest to the required capacity. The optimal number of logical processors to be used is kept within a book boundary wherever possible, thereby preventing L2 cache misses that would have occurred when the dispatcher could dispatch work wherever a processor might be available.

Figure 2-21 on page 140 shows, on the left side, that with HIPERDISPATCH=NO, the physical processors run different tasks from different logical partitions. On the right side of the figure, with the option YES, the physical processor 1 only switches between task A and task C from the same logical partition.
2.22 HiperDispatch considerations

- Logical CPUs (target 4) assigned a node with consideration for book boundaries based on PR/SM guidance
- z/OS uses this to assign logical processors to nodes and work to those nodes
- Periodic rebalancing of task assignments
- Assign work to the minimum number of logicals needed to use weight - remaining logical CPUs to use white space
- Requires "tightening up" of WLM policies for important work
- Single HIPERDISPATCH=YES z/OS IEAOPTxx parameter dynamically activates HiperDispatch (full S/W and H/W collaboration) without IPL
- With HIPERDISPATCH=YES, IRD management of LPs is turned OFF

HiperDispatch considerations

The HiperDispatch function is dynamically activated through HIPERDISPATCH=YES in the IEAOPTxx member in parmlib. In this case, the Vary Logical CPU Management function of IRD is automatically switched off.

The z/OS dispatcher tries to use a minimum number of active logical processors in an LPAR. This is done to increase the probability of having the same logical processor execute the same dispatchable unit (TCB/SRB). (In order to dispatch a TCP/SRB in a logical processor, the z/OS dispatcher must be running in that logical processor.) The HiperDispatch function engages in a constant communication between the LPAR code and the z/OS dispatcher.

This feature is available with z/OS V1R9, and for z/OS V1R7 with a zIIP Web deliverable.
2.23 Central storage design

Central storage, as previously mentioned, is located in the four books. Each basic memory element is the Dual in-line Memory Module (DIMM). The DIMM is a series of random access memory integrated circuits (a bit is implemented through a flip-flop circuitry).

Memory in a book is organized in two logical pairs:
- Logical pair 0 has two memory control units (MCU 0 and MCU1). Each MCU controls four groups of three DIMMs.
- Logical pair 1 has two memory control units (MCU 2 and MCU 3). Each MCU controls four groups of three DIMMs.

Within a logical MCU pair, the DIMM size (4 GB or 8 GB) must be the same. In addition, the total memory capacity for each logical MCU pair must be the same. MCU implements a non-associative memory concept; that is, the Storage Controller delivers an address and the MCU returns the contents of central storage associated with that address.

The maximum capacity per book: 8 GB x 12 x 4 = 384 GB. Note that memory sizes in each book do not have to be similar; different books may contain different amounts of memory.

Memory can be purchased in increments of 16 GB, up to a total size of 256 GB. From 256 GB, the increment size doubles to 32 GB until 512 GB. From 512 GB to 944 GB, the increment is 48 GB. Beyond that, up to 1520 GB, a 64 GB increment is used.
2.24 Addresses and addresses

- Virtual address (CP) to a Real Address (DAT)
- Real Address (CP) to a z/Architecture Absolute Address (Prefixing)
- z/Architecture absolute address to z10 EC Absolute Address (LPAR)
- z10 EC Absolute Address (LPAR) to z10 EC Book/memory physical address (Storage Controller)

Addresses and addresses

During program execution, the instruction address is virtual in the PSW and its operand addresses are also virtual in the base registers plus displacement. To access the instruction and the operands in caches or central storage, these virtual addresses must be translated and converted into other types of addresses. Figure 2-24 shows the stages during that process:

- Virtual address to real address
  Executed by DAT through the use of tables and TLB. The operating system sets up the translate controls (CRs) and translate tables.

- Real address to a z/Architecture absolute address
  Executed by the hardware function prefixing using the PVR register. The absolute address allows each processor to have a unique 8 KB PSA. Refer to z/Architecture Principles of Operations, SA22-7832, to learn about the concept of prefixing. Prefixing proceeds as follows:
  - If the real address is equal to 0KB - 8 KB (PSA) - forward prefix
  - If the real address is equal to the Prefix Register - reverse prefix
  - If the real address is not equal to the PSA address, or not equal to the prefix register address - then no prefix is used and the real address is an absolute address

- z/Architecture absolute address to z10 EC absolute address
Executed by the LPAR, as follows:

- LPAR sets up the logical partition relocate offset at image activation time.
- Adds a “relocate” offset to a z/Architecture absolute address. If the LP is the first in central storage, its relocate offset is the size of the HSA. If the logical partition is second in central storage, its relocate address is the sum of HSA size plus the first LP size.

▲ z10 EC absolute address to z10 EC book/memory physical address

The z10 EC absolute address is used to access the z10 EC Configuration Array to get the z10 EC book’s memory physical address. This z10 EC book’s memory physical address indicates the book number, the logical pair number, and the MCU requested.
2.25 Hardware system area (HSA)

- 16 GB of HSA in any z10 EC model at power-on Reset
- Eliminates the installation pre-planning task for HSA configuration and eventual expansion size
- This size is able to contain:
  - 4 CSSs
  - 15 LPs in each CSS (total of 60 LPs)
  - Subchannel set-0 with 63.75k devices in each CSS
  - Subchannel set-1 with 64k devices in each CSS
- All these are designed to be activated and used with dynamic I/O changes

**Figure 2-25  Hardware system area (HSA)**

**Hardware system area (HSA)**

HSA is built along a power-on Reset (POR) HMC operation. It contains the description of the channel subsystems, logical partitions, and I/O configuration. On top of that, it contains the LPAR LIC code and a copy of the CFCC operating system LIC code. It is always located in the central storage of the first book.

In z10 EC always occupies 16 GB of central storage. As a result, in a z10 EC, the installation planning for the HSA size is eliminated. Also, the pre-planning for HSA expansion for reconfiguration is eliminated because HCD/IOCP will, via the POR process, always reserve the maximum size (16 GB) for the maximum I/O configuration possible. Note that there is room in the HSA for 7665 K (127.75 K x 60 LPs) subchannels (or UCWs).

**Note:** HSA central storage is not included in the size of central storage that you purchase.
2.26 Large page (1 M) support

- Due to the number of getmained pages explosion:
  - Applications suffer a significant performance penalty because of TLB misses
  - Not allowed to increase the number of TLB entries because performance constraints

- Solution:
  - Decrease the number of pages making them bigger. Instead of 4 K, we have 1 M. Long running memory access intensive applications benefit
  - Optionally requested in IEASYSnn member with the following parameter: LFAREA=nn% | nnM | nnG
  - Large pages are treated as fixed pages and only available above 2 GB

Figure 2-26   Large page (1M) support

Large page (1M) support
The z/Architecture virtual storage implementation defines:
- Pages of 4 K addresses
- Segments of 1 M addresses
- Regions of 2 G addresses

Virtual addresses are sliced into page, segment, and region numbers to be translated by the dynamic address translation (DAT) through the use of page tables, segment tables, and up to three region tables.

The translation lookaside buffer (TLB) is a fast memory located in each PU. In the z10 EC, it has 512 entries. Each entry contains the real address associated with the most referenced 512 pages. The TLB accelerates consistently the DAT translation performance because with a TLB hit, DAT does not need to go to central storage in order to access the tables for translation.

However, the expected use of virtual addresses above the bar (2 G) is causing an explosion of pages. On the other hand, TLB sizes have remained relatively small due to low access time requirements and hardware space limitations.

TLB coverage today represents a much smaller fraction of an application’s working set size pages, thus leading to a larger number of TLB misses. Applications can suffer a significant
performance penalty resulting from an increased number of TLB misses, as well as the increased cost of each TLB miss.

So, if we are not allowed to increase the number of TLB entries, and virtual storage usage is exploding, the solution is to decrease the number of pages by making them bigger. That is, instead of 4 K, we have 1 M. Using large pages ensures that the TLB better represents the working set, and suffers fewer misses by allowing a single TLB entry to cover more address translations.

Exploiters of large pages are better represented in the TLB and are expected to perform better: long-running memory access-intensive applications particularly benefit. Short processes with small working sets see little or no improvement. The decision to use large pages must be based on the knowledge you gain from measuring memory usage and page translation overhead for a specific workload.

The large page size of 1 M addresses is optionally requested in the z/OS IEASYSn member of SYS1.PARMLIB with the following parameter:

\[ \text{LFAREA=nn\% | nnM | nnG}. \]

Note that after the large page size is selected, it cannot be dynamically changed without an IPL. Large pages are treated as fixed pages and are never paged out. They are only available for 64-bit virtual private storage such as virtual memory located above 2 GB.
2.27 Connecting PU cage with I/O cages

Comparing z9 EC with z10 EC I/O interconnect technology

The interconnect technology used in z9 EC was eSTI (self-time interconnect) with data rates of 2.7 GB/sec. The eSTI cable connected two traffic processors: the MBA in the PU cage and the STI-MP in the I/O cage. Each book has 8 MBAs (in fanout card) and each MBA connects to two eSTIs. One STI-MP controls the I/O flow of an I/O cage domain. A domain is formed by four I/O cards in the I/O cage.

As I/O cards continue to support higher data transfer rates to the devices, the connection between the I/O cards and the CEC cage needs to provide a higher data rate, as well. The connectivity to the I/O cages (I/O domains) in the System z10 is implemented by InfiniBand technology, which provides a data rate of 6 GB/sec. Refer to “InfiniBand interconnect technology” on page 153 for more information about this topic.

In the z10 EC I/O cage, the InfiniBand multiplexer (IFB-MP) card replaces the self-timed interconnect multiplexer (STI-MP) card present in the z9. The z10 EC supports a combination of old technology memory bus adapter (MBA) and host channel adapter (HCA) InfiniBand fanout cards on the CEC cage. The MBA fanout cards are used exclusively for ICB-4 Coupling Facility links.

With the System z10, there are two types of HCAs in one book:
- The HCA2-C fanout connects via an IFB copper cable to an IFB-MP (InfiniBand - Multiplex) card, installed in the I/O cages; see Figure 2-27.
The HCA2-O fanout connects via a CF link optical cable (external) to another z10 EC server.

Each fanout has two ports to connect either a copper cable IFB (internal) or optical. There are up to eight HCA2-C fanout cards per book. However, the maximum number of fanout cards per server is 24 with 2 ports each.

Then, multiplying 24 x 2 x 6 MB/sec, we have a 288 GB/sec of total z10 EC aggregate data rate.

I/O connectivity

The z10 EC has Host Channel Adapter (HCA) fanouts residing in the front of the book. There are unique HCA fanouts supporting I/O features and Parallel Sysplex coupling links in the I/O cage.

The z10 EC generation of the I/O platform is intended to provide significant performance improvement over the current I/O platform used for FICON Express4, OSA-Express2, and Crypto Express2. It will be the primary platform to support future high-bandwidth requirements for FICON/Fibre Channel, Open Systems adaptors, and Crypto.
2.28 Detailed connectivity

The optical cable is used for coupling link external connectivity, while the copper cable is used for I/O cage connectivity. There are three different fanouts (Host Channel Adapter - HCA) based on InfiniBand technology:

- HCA1-O (also called MBA as in the z9 server): connecting through CF links within system z9, optical, up to 3 GB/sec, up to 150 meters. This is not used by z10 EC.
- HCA2-O: connecting through CF links with system:
  - z10: optical, up to 6 GB/sec
  - z9: more than 10 meters distance, optical, up to 3 GB/sec.
  The rate is negotiable.
- HCA2-C: connecting internally with I/O cages, copper, up to 6 GB/sec.

Connectivity summary

The fiber cables are industry standard OM3 (2000 MHz-km) 50 micron multimode optical cables with Multi-Fiber Push-On (MPO) connectors. The maximum cable length is 150 meters (492 feet).

Each fiber supports a link rate of 6 GBps (12x IB-DDR) if connected to a z10 EC server or 3 GBps (12x IB-SDR) when connected to a System z9 server. The link rate is auto-negotiated to the highest common rate.
2.29 HCA and I/O card connections

Figure 2-29 shows:

- HCA2-C adapters connected through copper cables with IFB-MP located in one I/O slot in the I/O cage. The rate in this case is 6.0 GB/sec. This IFB-MP handles and manages the flow of four I/O cards constituting a domain.

- HCA2-O adapters connected through optic fiber cables named Parallel Sysplex using Infiniband (PSIFB) with HCA2-O located in other z10 EC server. The rate in this case is 6.0 GB/sec.

- MBA adapters connected through copper cables with an ICB-4. It is possible to hot swap ICB-4 and InfiniBand hub cards.

  New ICB-4 cables are needed for z10 EC.

The MBA fanout provides coupling links (ICB-4) to either z10 EC servers or z9, z990, and z890 servers. This allows you to use the z10 EC and earlier servers in the same Parallel Sysplex.

MBA fanouts are only for ICB-4 coupling links and cannot be used for any other purpose. When upgrading to a z10 EC from a System z9 or z990 with ICB-4 coupling links, new ICB copper cables are required because connector types used in the z10 EC are different from the ones used for z9 and z990.

The ICB-4 feature cannot be ordered on a Model E64 server.
2.30 InfiniBand interconnect technology

InfiniBand is a powerful interconnect technology designed to deliver I/O connectivity for large server and network infrastructures. It is supported by all major server vendors as a means to deliver the next generation I/O interconnect standard for servers.

System z servers are able to do more and more work and in order to keep up with the increasing performance, it becomes necessary to introduce an interconnect architecture that is able to scale with the increasing performance of the platform to satisfy the I/O interconnect requirements that go along with it. InfiniBand is a powerful interconnect architecture that is better able to scale with increasing processor speeds.

In 1999, two competing input/output (I/O) standards called Future I/O (developed by Compaq, IBM and Hewlett-Packard) and Next Generation I/O (developed by Intel, Microsoft and Sun) merged into a unified I/O standard called InfiniBand. The InfiniBand Trade Association (IBTA) is the organization that maintains the InfiniBand specification and is led by a steering committee manned by members of the earlier mentioned corporations.

InfiniBand is an industry-standard specification that defines an input/output architecture used to interconnect servers, communications infrastructure equipment, storage and embedded systems. InfiniBand is a true fabric architecture that leverages switched, point-to-point channels with data transfers up to 12 MB per second, both in chassis backplane applications as well as through external copper and optical fiber connections. InfiniBand is a pervasive, low-latency, high-bandwidth interconnect which requires low processing overhead and is ideal...
for carrying multiple traffic types (clustering, communications, storage, management) over a single connection.

**Advantages of InfiniBand**
Following are some advantages of using InfiniBand:

**Superior performance.** InfiniBand provides superior latency performance and products supporting up to 12 MB/sec connections.

**Reduced complexity.** InfiniBand allows for the consolidation of multiple I/Os on a single cable or backplane interconnect, which is critical for blade servers, data center computers and storage clusters, and embedded systems.

**Highest interconnect efficiency.** InfiniBand was developed to provide efficient scalability of multiple systems. InfiniBand provides communication processing functions in hardware, thus relieving the processor of this task, and it enables full resource utilization of each node added to the cluster.

**Reliable and stable connections.** InfiniBand provides reliable end-to-end data connections and defines this capability to be implemented in hardware.

Figure 2-30 on page 153 shows the InfiniBand communications stack. Several types of architected transactions can be used to execute a transaction with another user. Work is posted on the appropriate queue, and the channel adapter executes the operation.

In case of a send operation, the channel adapter interprets the type of work, creates a message, segments it (if needed) into multiple packets, adds the routing information, and sends the packets to a port. Port logic is now responsible for sending the packet (or packets) across the link through the fabric to its destination. When the packet arrives, the port logic validates the packet, and the channel adapter puts it on the queue and executes it. If requested, the channel adapter creates an acknowledgement and sends it back to its origin.
2.31 I/O cage

Figure 2-31 shows a physical view of the I/O cage. The z10 EC server can have up to three I/O cages to host the I/O and cryptographic cards required by a configuration. Each I/O cage has 28 I/O slots and supports up to seven I/O domains (from A to G). Each I/O domain is made up of up to four I/O slots and a IFB-MP, as shown.

Each I/O domain requires one IFB-MP card. All I/O cards within an I/O domain are connected to its IFB-MP card via the back plane board. A full I/O cage requires eight IFB-MP cards, which are half-high cards, using four slots.

If one I/O domain is fully populated with ESCON cards (each with 15 active ports and one spare per card), then up to 60 (four cards x 15 ports) ESCON channels can be installed and used. An I/O cage that has six domains fully populated with ESCON cards will have 360 (60 x 6 domains) ESCON channels.

Each IFB-MP card is connected to an HCA2-C port located in the book. Because each IFB-MP card requires one port, up to eight ports are required to support one I/O cage.

The configuration process selects which slots are used for I/O cards and supplies the appropriate number of I/O cages and IFB cables, either for a new build server, or for a server upgrade.

A full I/O cage requires four IFB-MP cards (from A to G), which are half-high cards, using three and a half slots. I/O cards can be installed or replaced concurrently. The I/O cards
contain the channel where the I/O logic is executed and the I/O ports to connect to the external devices, networks or to other servers.

All channels of an I/O card are served by the same SAP.

In addition, two Distributed Converter Assembly-Cage Controller (DCA-CC) cards plug into the I/O cage.

The maximum number of channels in the server by type of channel is listed here:

- Up to 1024 ESCON channels (up to 960 on the model E12)
- Up to 120 FICON Express channels (when carried forward on upgrade only)
- Up to 336 FICON Express2 channels (when carried forward on upgrade only)
- Up to 336 FICON Express4 channels
- Up to 24 OSA-Express3 features
- Up to 24 OSA-Express2 features
- Up to 48 ISC-3 coupling links
- Up to 16 ICB-4 coupling links (up to 32 with RPQ 8P2337)
- Up to 32 InfiniBand coupling links
- Two External Time Reference (ETR) connections

**z10 I/O cage**

The z10 EC has a minimum of one CEC cage and one I/O cage in the A frame. The Z frame can accommodate an additional two I/O cages, making a total of three for the entire system.

One I/O cage can accommodate the following card types:

- Up to eight Crypto Express2
- Up to 28 FICON Express4, FICON Express2, or FICON Express
- Up to 24 OSA-Express2 and OSA-Express3
- Up to 28 ESCON

It is possible to populate the 28 I/O slots in one I/O cage with any mix of these cards.
2.32 The I/O data flow

I/O data flow for a read - example

Here we follow a data flow experienced by a 4 KB DASD FICON read, through all the paths that the data might travel:

1. From a disk track to the DASD controller cache - and it was a cache miss.
2. From the DASD controller cache to DASD controller host adapter buffer.
3. From the DASD controller host adapter buffer to the FICON switch director port buffer (moved within FICON data frames).
4. From the FICON director port buffer (moved within FICON data frames) to the channel I/O port buffer in one I/O card in one I/O cage.
5. From a channel I/O port buffer in one I/O card in an I/O cage, to an I/O port in the IFB-MP card.
6. From an I/O port in the IFB-MP card to an HCA2-O buffer.
7. From an HCA2-O buffer to the L2 cache.

As shown in Figure 2-32, using a tree as an analogy for the z10 EC, the “top leaves” (I/O channels) connect to a little branch, which connect to a larger branch to the trunk. All I/O channels in one I/O card fork in the I/O card port. Four I/O card ports fork in an IFB-MP. Two IFB-MPs fork in a HCA2-O (there are two IFB-MP links per HCA2-O fanout card). Eight HCA2-Os fork in one L2-Cache. In all forks, traffic controllers allow, serially, all the confluent flows (refer to Figure 2-27 on page 149).
2.33 Redundant I/O Interconnect

Redundant I/O Interconnect is a feature of the z10 EC server. It is accomplished by the facilities of the IFB-MP card.

Each IFB-MP card is connected to an IFB jack located in the HCA2-C fanout card of a book. IFB-MP cards are half-high cards and are interconnected, thus allowing redundant I/O interconnect. If the IFB connection coming from a book ceases to function (such as when a book is removed), the interconnect path is used. Figure 2-33 shows a conceptual view of how Redundant I/O Interconnect is accomplished.

Normally, book 0 HCA2-C connects to the IFB-MP (A) card and services domain 0 I/O connections (slots 01, 03, 06, and 08). Book 1 HCA2-C/IFB connects to the IFB-MP (B) card and services domain 1 (slots 02, 04, 07, and 09).

If book 1 is removed, or if the connections from book 1 to the cage are removed, connectivity to domain 1 is maintained by guiding the I/O to domain 1 through the interconnect between IFB-MP (A) and IFB-MP (B).
2.34 z10 EC I/O features supported

Figure 2-34 I/O features supported

<table>
<thead>
<tr>
<th>I/O feature</th>
<th>Feature codes</th>
<th>Number of</th>
<th>Max. number of</th>
<th>CHPID</th>
<th>CHPID definition</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESSON</td>
<td>5325 5324</td>
<td>16 (16 ports)</td>
<td>1024</td>
<td>69</td>
<td>Yes</td>
<td>CNC, OVC, CTC, CBY</td>
</tr>
<tr>
<td>FICON Express</td>
<td>2319/2320</td>
<td>2</td>
<td>1200</td>
<td>69</td>
<td>Yes</td>
<td>FG, FCP, PCC</td>
</tr>
<tr>
<td>FICON Express2 2319/3220</td>
<td>4</td>
<td>4</td>
<td>356</td>
<td>84</td>
<td>Yes</td>
<td>FG, FCP</td>
</tr>
<tr>
<td>FICON Express4 3244/3222</td>
<td>4</td>
<td>4</td>
<td>356</td>
<td>84</td>
<td>Yes</td>
<td>FG, FCP</td>
</tr>
<tr>
<td>OSA Express2 GSE 3364/3965</td>
<td>2</td>
<td>2</td>
<td>48</td>
<td>24</td>
<td>Yes</td>
<td>OSD, OSN</td>
</tr>
<tr>
<td>OSA Express2 GSE 3358</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>24</td>
<td>Yes</td>
<td>OSD</td>
</tr>
<tr>
<td>OSA Express2 GSE 3855</td>
<td>2</td>
<td>2</td>
<td>48</td>
<td>24</td>
<td>Yes</td>
<td>OSE, OSH, OGC, OSHN</td>
</tr>
<tr>
<td>OSA Express3 GSE 3970</td>
<td>2</td>
<td>2</td>
<td>48</td>
<td>24</td>
<td>Yes</td>
<td>OSE</td>
</tr>
<tr>
<td>ISC-3 3Gbps (1port)</td>
<td>0217,0214,0219</td>
<td>2 / ISC-D</td>
<td>43</td>
<td>12</td>
<td>Yes</td>
<td>CFP</td>
</tr>
<tr>
<td>ISC-3 1Gbps (2ports)</td>
<td>83F2197</td>
<td>2 / ISC-D</td>
<td>43</td>
<td>12</td>
<td>Yes</td>
<td>CFP</td>
</tr>
<tr>
<td>ICP 4</td>
<td>2002</td>
<td>2</td>
<td>16</td>
<td>-</td>
<td>Yes</td>
<td>CEP</td>
</tr>
<tr>
<td>ICP</td>
<td>-</td>
<td>2</td>
<td>32</td>
<td>-</td>
<td>No</td>
<td>CEP</td>
</tr>
</tbody>
</table>

Figure 2-34 shows all the types of channels and their limits in a z10 EC server.

The InfiniBand coupling (IFB), which is not shown in the figure, has a maximum of 32 ports and does not have a PCHID associated to it. Refer to “Physical channel IDs (PCHID)” on page 253 for more information on PCHIDs.

The notes in the last column of Figure 2-34 refer to the following points:

- Notes 1,2 provide more detailed information about the two listed feature codes.
- Note 3 indicates that only FICON Express-4 (400 MB/sec) are natively available. FICON Express-2 and FICON Express are only supported, if carried over, on an upgrade.
- Note 4 indicates that the maximum number of combined IB, ISC-3 and IC coupling links is 64.
2.35 16-port ESCON channel card

Figure 2-35  A 16-port ESCON channel card

Z9 EC 16-port ESCON feature
The 15 active ports on each 16-port ESCON feature are activated in groups of four ports via Licensed Internal Code-Control Code (LIC-CC) by using the ESCON channel port feature (FC 2324).

The first group of four ESCON ports requires two 16-port ESCON features. After the first pair of ESCON cards is fully allocated (by seven ESCON ports groups, using 28 ports), single cards are used for additional ESCON ports groups.

Ports are activated equally across all installed 16-port ESCON features for high availability. In most cases, the number of physically installed channels is greater than the number of active channels that are LIC-CC enabled. This is not only because the last ESCON port (J15) of every 16-port ESCON channel card is a spare, but also because several physically installed channels are typically inactive (LIC-CC protected). These inactive channel ports are available to satisfy future channel adds.

Z10 ESCON Ports
Z10 EC supports the MIDAW facility, previously exclusive to the Z9. The System Z9 I/O subsystem supports a facility for indirect addressing, Modified Indirect Data Address Word (MIDAW) facility, for both ESCON and FICON channels. The use of the MIDAW facility by applications that currently use data chaining may result in improved channel throughput in FICON environments. A maximum of 1024 ESCON ports can be activated on a Z10 EC server. This maximum requires 69 16-port ESCON channel cards to be installed.
2.36  FICON features and extended distance

- Only FICON Express-4 (400 MB/sec) - FICON Express-2 and FICON Express are only available if carried over, on an upgrade

- Express4 features:
  - Long wave (LX) 9 micron single mode fiber optic cables. Two options for unrepeated distances of:
    - Up to 4 kilometers (2.5 miles)
    - Up to 10 kilometers (6.2 miles).
  - Short wavelength (SX) - For short distances, uses 50 or 62.5 micron multimode fiber optic cables
  - FICON Extended Distance, new protocol for Information Unit (IU) pacing

FICON features

The FICON Express4 features are available in long wavelength (LX) and short wavelength (SX). For customers exploiting LX, there are two options available for unrepeated distances of up to 4 kilometers (2.5 miles) or up to 10 kilometers (6.2 miles). Both LX features use 9 micron single mode fiber optic cables.

The SX feature uses 50 or 62.5 micron multi-mode fiber optic cables.

Each FICON Express4 feature (I/O card) has 4 independent channels (ports) and can be configured to carry native FICON traffic or Fibre Channel (SCSI) traffic. LX and SX cannot be intermixed on a single feature, and the receiving devices must correspond to the appropriate LX or SX feature. The maximum number of FICON Express4 features is 84 using three I/O cages.

An enhancement to the industry standard FICON architecture (FC-SB-3) helps avoid degradation of performance at extended distances by implementing a new protocol for “persistent” Information Unit - IU (also named frames) pacing.

Control units that exploit the enhancement to the architecture can increase the pacing count (the number of IUs allowed to be in flight from channel to control unit). Extended Distance FICON also allows the channel to “remember” the last pacing update for use on subsequent operations to help avoid degradation of performance at the start of each new operation.
Improved IU pacing can help to optimize the utilization of the link (for example, to help keep a 400 MB/sec link fully utilized at 50 Km) and provide increased distance between servers and control units. The requirements for channel extension equipment are simplified with the increased number of commands in flight. This may benefit z/OS Global Mirror (Extended Remote Copy, XRC) applications because the channel extension kit is no longer required to simulate (or spoof) specific channel commands. Simplifying the channel extension requirements may help reduce the total cost of ownership of end-to-end solutions. Extended distance FICON is transparent to operating systems, and it applies to all the FICON Express4 and FICON Express2 features carrying native FICON traffic (CHPID type FC).

For exploitation, the control unit must support the new IU pacing protocol. The DS8000 series License Machine Code (LMC) level 5.3.1xx.xx supports such protocol. The channel defaults to current pacing values when operating with control units which cannot exploit extended distance FICON.

**z10 FICON support**

The z10 EC generation of the I/O platform is intended to provide significant performance improvement over the current I/O platform used for FICON Express4, OSA-Express2, and Crypto Express2. It will be the primary platform to support future high-bandwidth requirements for FICON/Fibre Channel, Open Systems adaptors, and Crypto.

As part of the high availability offered by z10 EC, FICON Cascaded Director continues to be supported. Cascaded support is important for disaster recovery and business continuity solutions. It can provide high availability and extended distance connectivity, and has the potential for fiber infrastructure cost savings by reducing the number of channels for interconnecting the two sites.

With high bandwidth, the z10 EC introduces InfiniBand as the internal interconnect protocol to drive ESCON and FICON channels, OSA-Express2 and OSA-Express3 ports, and ISC-3 coupling links. As a connection protocol, InfiniBand supports ICB-4 links and InfiniBand coupling (PSIFB) with up to 6 GBps link rate.

With wide connectivity, the z10 EC can be connected to an extensive range of interfaces such as Gigabit Ethernet (GbE), FICON, ESCON, and coupling links.
2.37 Features in z10 and z196

- Logical channel subsystem
- Channel CHIPD and PCHID
- MF ID and LP ID
- HCD/IOCP
- Channel Availability Features
- MIDAW
- DASD Extended Format
- Crypto Features
- Capacity Upgrades
- Capacity Provisioning

Features in z10 and z196
All the features and functions listed in Figure 2-37 are available for z10 and z196. To avoid duplication they are described in the z196 chapter only.
2.38 z10 EC new features

Figure 2-38 shows the most important hardware and LIC features announced with the z10 EC servers. With a few exceptions, these features are covered in this chapter.

HiperSockets Multiple Write Facility

For increased performance, HiperSockets performance has been enhanced to allow for the streaming of bulk data over a HiperSockets link between logical partitions (LPs). The receiving LP can now process a much larger amount of data per I/O interrupt.

This enhancement is transparent to the operating system in the receiving LPAR. HiperSockets Multiple Write Facility is designed to reduce processor utilization of the sending LPAR. HiperSockets Multiple Write Facility on the z10 EC requires at a minimum z/OS 1.9 with PTFs.
Introducing the IBM zEnterprise

Some of the key requirements today in commercial environment are the need to maximize return on investments (ROI) by deploying resources designed to drive efficiencies and economies of scale; managing growth through resources that can scale to meet changing business demands; reducing risk by minimizing the threat of lost productivity through downtime or security breaches; reduce complexity by reversing the trend of server proliferation (reaching now many thousands in many companies); and enabling business innovation by investing resources that can help protect existing investments, while also enabling new technologies that enable business transformation.

The IBM zEnterprise System delivers a world-class enterprise server designed to meet these business needs. The zEnterprise System is the first of its kind. It was designed to help overcome problems in today’s IT infrastructures and provide a foundation for the future. The zEnterprise System represents both a revolution and an evolution of mainframe technology. IBM is taking a bold step by integrating heterogeneous platforms under the well-proven System z hardware management capabilities, while extending System z qualities of service to those platforms.

Recent decades have witnessed an explosion in applications, architectures, and platforms. A lot of experimentation has occurred in the marketplace. With the generalized availability of the Internet and the appearance of commodity hardware and software, various patterns have emerged that have gained center stage.

Multi-tier application architectures and their deployment on heterogeneous infrastructures are common today. But what is uncommon is the infrastructure setup needed to provide the high qualities of service required by mission critical applications.

Creating and maintaining these high-level qualities of service while using a large collection of distributed components takes a great amount of knowledge and effort. It implies acquiring and installing extra equipment and software to ensure availability, integrity, security, monitoring, and managing. Lots of additional manpower is required to configure, administer, troubleshoot, and tune such a complex set of separate and diverse environments. Due to platform functional differences, the resulting infrastructure will not be uniform, regarding those qualities of service or serviceability.
Careful engineering of the application's various tiers is required to provide the robustness, scaling, consistent response time, and other characteristics demanded by the users and lines of business. These infrastructures do not scale well. What is a feasible setup with a few servers becomes difficult to handle with dozens and a nightmare with hundreds. When it is doable it is expensive. Often, by the end of the distributed equipment's life cycle, its residual value is nil, requiring new acquisitions, software licenses, and re-certification. It is like going back to square one. In today’s resource-constrained environments, there is a better way.

To complete this picture on the technology side, performance gains from increasing the frequency of chips are becoming smaller. Thus, special-purpose compute acceleration will be required for greater levels of workload performance and scalability.

Workloads have changed. Now many applications, including mission-critical ones, are implemented in heterogeneous infrastructures and the System z design has adapted to this change. The zEnterprise System design can simultaneously support a large number of diverse workloads while providing the highest qualities of service.

The zEnterprise System is following an evolutionary path that directly addresses those infrastructure problems. Over time, it will provide increasingly complete answers to the smart infrastructure requirements. zEnterprise, with its heterogeneous platform management capabilities, already provides many of these answers, offering great value in a scalable solution that integrates and simplifies hardware and firmware management and support, as well as the definition and management of a network of virtualized servers, across multiple heterogeneous platforms.

The zEnterprise has three major components: z196 serve, zBX Blade server and a piece of software named Unified Resource Manager or also called zManager.

The z196 delivers scalability and granularity to meet the needs of medium-sized enterprises, while also satisfying the requirements of large enterprises having large-scale, mission-critical transaction and data processing requirements. zEnterprise expands the subcapacity settings offer with three subcapacity levels for the first 15 processors, giving a total of 125 distinct capacity settings in the system, and providing for a range of over 1:200 in processing power.

The zEnterprise continues to offer all the specialty engines (such as zIIP, zAAP, IFL, ICF, SAP) available with System z10, to help lower the cost of ownership. Also, specialty engines continue to help users expand the use of the mainframe for a broad set of applications.

Combining the maximum capacity of zBX with the maximum capacity of z196 we may reach the astronomical capacity of 1-tera instructions per second (TIPS).

The zBX is a cheap solution to run Unix applications from software houses where there is no support for running such as zLinux. Another zBX role is to host z offloaded optimizer functions such as Dataware House processing.

The IBM holistic approach to System z design includes hardware, software, and procedures. It takes into account a wide range of factors, including compatibility and investment protection, thus ensuring a tighter fit with the IT requirements of the entire enterprise.

As environmental concerns raise the focus on energy consumption, the zEnterprise is designed to reduce energy usage and save floor space when used to consolidate x86 servers.

In this chapter we cover the z196 in detail.
3.1 zEnterprise overview

Figure 3-1  A zEnterprise picture

**zEnterprise components**
A zEnterprise consists of three major components:

- The zEnterprise 196 central processor complex (CPC) is also named z196. This name comes from the fact that in the maximum configuration the z196 may have 96 Processor Units (engines). The z196 machine type is 2817 (the z10 is 2097).
- The zEnterprise BladeCenter Extension is also named zBX Model 002.
- The zEnterprise Unified Resource Manager Licensed Internal Code (LIC) is IBM software not perceived or even accessed by the customer.

The zEnterprise’s major objectives are:

- To integrate, monitor, and manage the heterogeneous infrastructure resources as a logical, single, virtualized system.
- To manage the infrastructure resources in accordance with specified business service level objectives associated with the business service workloads.
- To provide dynamic deployment and management of virtual server images and virtualized appliances in a service optimized infrastructure.

**zEnterprise 196 CPC (z196)**
The zEnterprise 196 CPC is the successor of the z10EC CPC. It has a newly designed quad-core chip, clocking at 5.2 GHz. The z196 can be configured with up to 80 processors
running concurrent production tasks with up to 3 TB of memory. It offers hot pluggable I/O drawers that complement the z9 introduced I/O cages, and continues the utilization of advanced technologies such as InfiniBand.

The z196 goes beyond previous designs while continuing to enhance the traditional mainframe qualities, delivering unprecedented performance and capacity growth. The z196 is a well-balanced general-purpose server that is equally at ease with compute-intensive and I/O-intensive workloads. It has two option concerned with cooling: water and air. IBM does recommend water cooling mainly for reasons of saving energy.

It runs all the mainframe operating systems as such: MVS, VSE, zLINUX, z/VM, z/TPF and CFCC (for coupling facility).

**zEnterprise BladeCenter Extension**

The integration of heterogeneous platforms is based on the IBM zEnterprise BladeCenter Extension (zBX) Model 002, which houses general purpose blades as well as blades for specialized solutions. The big differential is to use the system management functions of the mainframe to manage the distributed environment.

A blade server is a highly compacted version of a server with processors, memory, operating system disks and I/O channels, all designed to slide into standard industry racks. They require less space than the traditional implementation.

The major reasons for the blades in the zEnterprise are:

- To incorporate additional “specialty engines” to host applications on their native processor architectures.
- To consolidate and manage a multiple-tier, heterogeneous infrastructure with reduced complexity and lower cost.
- To enable better application integration with IBM System z transaction processing, messaging, and data serving capabilities.

The processor technology is Power-7. The communication with z196 is through a closed OSA-Express3 (1000BASE-T) network and is located up to 26 meters from z196. These blades can:

- Run independently AIX® (SOD Linux on x) as normal blades but managed by zEnterprise Unified Resource Manager.
- Offload transaction functions to special performance software blades (such as the IBM Smart Analytical Optimizer). Here, we have the same classic idea of offloading functions from CPU to other processors. Some other examples can be: crypto, data compress, SAP, channels, and Vector processors.

**zEnterprise Unified Resource Manager (zManager)**

Another key element is a zEnterprise a LIC loaded in the Hardware Management Console (HMC) and in the Support Element. It is not shown in Figure 3-1 on page 167, but its functions are described in the circle slices.

Its functions are grouped as follows:

- Operating (defining and managing) virtual environments including hypervisors such as z/VM, LPAR, andPowerVM (PVM). This includes the automatic definition as well as the discovery of I/O and other hardware components across z196 and zBX, and the definition and management of LPs, virtual machines, virtual servers, and virtualized LANs.
- Receiving and applying corrections and upgrades to the Licensed Internal Code. Before zManager, the customer was in charge of applying such corrections at the blades.
- Performing temporary and definitive zEnterprise capacity upgrades.
- Managing and monitoring energy consumption.

**zEnterprise nodes and ensembles**

Together, z196, zBX (with up to 112 blades), and Unified Resource Manager constitute a node in a zEnterprise ensemble.

A zEnterprise ensemble can have a maximum of eight nodes, comprised of up to eight z196 servers and 896 blades (housed in eight zBXs). The ensemble has dedicated integrated networks for management and data and the Unified Resource Manager functions. Then, a zEnterprise ensemble is a collection of highly virtualized heterogeneous systems, managed as a single logical entity, where many diverse workloads can be deployed.

As this chapter progresses, we clarify the concepts covered in Figure 3-1 on page 167, such as: operations, energy, hypervisors, network, performance, and virtual servers.
3.2 Migrating Unix/Linux workloads back to the mainframe

Here we introduce the reasons that make a zEnterprise concept a good idea, that is, putting together z and p processors (RISC) under the same logical cover.

With the downsize revolution, many customers converted their CICS/Cobol applications to distributed platforms, mainly the ones based on UNIX. Currently many of those customers, due to the explosion of servers and the real high cost of distributing data and programs, are again considering the mainframe as their main server. However, they do not want to migrate back their business application home-made code and acquired products to CICS/Cobol.

In order to clarify such scenarios, we may say that a computer system (in a simplified way) is made of the following pieces or layers:

- Human skill
- Business application code
- Data
- Operating system and middleware code
- Hardware
Migration first scenario

If a customer is not dependent on the operating system code (UNIX or Linux) and on the hardware (RISC processors), then just the three top components of Figure 3-2 on page 170 are migrated:

- The human skill that is preserved by the use of the Korn Shell running under the z/OS USS component.
- The business application code, whose compatibility is preserved by the implementation of the Posix standard of Application Program Interfaces (API) set of standards. z/OS USS components mimic the UNIX/Linux behavior for any API call coming from the application code.
- The data is allocated in one HFS-like file accessed transparently by the application.

Migration second scenario

If a customer has dependencies on the operating system code (UNIX or Linux) but not with the hardware (RISC processors), then the four top components in the above mentioned picture are migrated:

- The human skill that is preserved by use of the Korn shell (or other shell) running under Linux.
- The business application code, whose compatibility is preserved because it is still running under Linux.
- The data is allocated in a real HFS file.
- The Linux code (if not Linux, a conversion from UNIX needs to be done) runs in a z196 logical partition usually, under z/VM. All the equivalent RISC instructions are executed by the z196 processors.

Migration third scenario

If a customer has hardware dependencies (as financial leasing contracts) running hardware (IBM RISC processors located in blades), then all the five components in the picture are migrated:

- The human skill that is preserved by use of the Korn shell (or other shell) running under Linux or UNIX.
- The business application code, whose compatibility is preserved because it is still running Linux or UNIX.
- The data is allocated in a real HFS file.
- The IBM blade is moved to zBX.
3.3 z196 numeric comparison

Here, we are starting to cover the details of technology, design, and architecture of the z196 family of processors, an element of the zEnterprise. Our first topic is the numeric comparison between a z196 and its predecessors. Figure 3-3 shows a four-axis comparison between five different processor lines: z900, z990, z9, z10 and z196. As you can see, the growth between z10 and z1960 is huge in three areas:

- Processor capacity index (PCI) is the new IBM name for MIPS. PCI per PU grew 1.31 times (1202 to 920). This number is calculated through the Large Systems Performance Reference (LSPR) measurements, that currently are using a Relative Nest Intensity (RNI) workloads. These workloads rely on processor MF information data in order to create different patterns of cache, memory and book accesses.

- The maximum number of PUs grew from 64 to 80 and the maximum number of any PU type grew from 77 to 96. PUs are the major z196 processors in charge of executing all types of code: operating system, middle ware, application.

- Central storage grew from 1.5 TB to 3 TB and the aggregate I/O data rate is the same at 288 GB/sec.

Central storage growth is proportionally larger than the other three resources. The justification is that in a heavy commercial data processing environment, central storage can really improve overall performance by drastically decreasing the I/O rate. On the other hand, zLinux workloads need lots of central storage to perform well and IBM is expecting (depending on the workload) almost one thousand of zLinux virtual machines in one large z196.
### Processor Units (PU) instances

- **Central processor (CPU or CP)**
- **Integrated facility for Linux (IFL)**
- **Integrated coupling facility (ICF)**
- **Application assist processor (zAAP)**
- **Integrated Information Processor (zIIP)**
- **System assisted processor (SAP)**
- **Spare**

**Processor unit (PU) instances or speciality engines**

As in previous z families of CPCs the z196 keeps the concept of speciality engines.

All PUs in a z196 are physically identical. However, at Power-on Reset, it is possible to have different types of PUs through a small LIC change (for the SAP PU, the change is larger), leading then to different instances. These instances are also called *specialty engines*. The major reason for such engines is to lower overall cost of computing in the mainframe platform. They can be ordered by customers and are known as *characterized PUs*. Following are the possible PU instances:

- **CPU**
  A CPU is a general purpose processor that is able to execute all the possible z196 running operating systems, as such: z/OS, Linux, z/VM, z/VSE, Coupling Facility Control Code (CFCC), and z/TPF. A CPU is also known as a CP.

- **IFL**
  This type of PU is only able to execute native Linux and Linux under z/VM. IFLs are less expensive than CPUs.

- **ICF**
  This type of PU is only able to execute CFCC operating system. The CFCC is loaded in a Coupling Facility LP from a copy in HSA; after this, the LP is activated and IPLed. ICFs are less expensive than CPUs.

- **zAAP**
  This type of PU only runs under z/OS, and is for the exclusive use of Java interpreter code (JVM) and DB2 9 XML parsing workloads. A zAAP is less expensive than a CPU, and does not increase the software price (based on...
consumed MSUs) because it does not produce MSUs. Starting on z/OS 1.12 all zAAP workloads migrate to zIIP and zAAPs start to be phased out.

**zIIP**

This type of PU is run in z/OS only, for eligible DB2 workloads such as DDF, business intelligence (BI). Also from ERP, CRM and IPSec (an open networking function used to create highly secure crypto connections between two points in an enterprise) workloads. A zIIP is less expensive than a CPU and does not increase the software price (based on consumed MSUs) because it does not produce MSUs. There is a limitation about using zIIP processors, that is, only a percentage of the candidate workload code can be executed on zIIP.

**SAP**

A System Assist Processor (SAP) is a PU that runs the Channel Subsystem LIC. SAP manages the starting of I/O operations required by operating systems running in all logical partitions. It frees z/OS (and the CPU) from this role, and is a “mainframe-unique”. z196 models have a variable number of standard SAPs configured.

**Spare**

A spare PU is a PU that is able to replace, automatically and transparently, any falling PU in the same book, or in a different book. There are at least two spares per z196 CPC.

**Simulation support**

z/VM guest virtual machines can create virtual specialty processors on processor models that support same types of specialty processors but do not necessarily have them installed. Virtual specialty processors are dispatched on real CPs. Simulating specialty processors provides a test platform for z/VM guests to exploit mixed-processor configurations. This allows users to assess the operational and CPU utilization implications of configuring a z/OS system with zIIP or zAAP processors without requiring the real specialty processor hardware.
3.5 z196 models

<table>
<thead>
<tr>
<th>Model</th>
<th>Books/PU</th>
<th>CPs</th>
<th>Standard SAPs</th>
<th>Spares</th>
</tr>
</thead>
<tbody>
<tr>
<td>M15</td>
<td>1/20</td>
<td>0-15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>M32</td>
<td>2/40</td>
<td>0-32</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>M49</td>
<td>3/60</td>
<td>0-49</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>M66</td>
<td>4/80</td>
<td>0-66</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>M80</td>
<td>4/96</td>
<td>0-80</td>
<td>14</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 3-5  z196 models**

**z196 models**

Here, we describe how many combinations of z196 PUs can be acquired. The z196 CPC hardware model nomenclature (also called Configs) is based on the number of PUs available for customer use in each CPC. These PUs are called characterized and they can take any PU personality chosen by the customer. The z196 has five models depending on the number and book capacity. A book may have 20 or 24 PUs. Refer to “z196 books” on page 183. There are a total of 126 capacity settings available:

**Model M15**: one book with 20 PUs, where three of them are standard SAPs and two are standard spare, then 15 characterized PUs. Refer to “z196 books” on page 183 to get more information about books.

**Model M32**: two books with 20 PUs each, where 6 of them are standard SAPs and 2 of them are standard spare.

**Model M49**: three books with 20 PUs each, where 9 of them are standard SAPs and 2 of them are standard spare.

**Model M66**: four books with 20 PUs each, where 12 of them are standard SAPs and 2 of them are standard spare.

**Model M80**: four books with 24 PUs each, where 14 of them are standard SAPs and two are standard spare. Upgrades from any z196 model to a model M80 is disruptive because this upgrade requires the replacement of four books.
3.6 Sub capacity models

Subcapacity models (also called sub-unis) are CPCs in which the processor speed is artificially downgraded due to financial and marketing considerations. In z196 there are a total of 45 subcapacity settings.

There are four capacity settings per processor. For example, the entry point (Model 401) is approximately 20% of a full speed processor (Model 701). However, all specialty engines (zAAP, zIIP, ICF, and IFL) continue to run at full speed. Subcapacity processors have all the availability of z196 features and functions. All processors must be the same capacity setting size within one z196.

Software model numbers
To recognize how many PUs are characterized as CPs (for software billing purposes), the store system information (STSI) instruction returns a value that can be seen as a model capacity identifier (MCI), which determines the number and speed of characterized CPs. MCI is also called a software model number. Characterization of a PU as an IFL, an ICF, a zAAP, or a zIIP is not reflected in the output of the STSI instruction, because these have no effect on software charging.

There is no affinity between the hardware model and the number of activated CPs. For example, it is possible to have a Model M66 (66 characterized PUs) but with only 13 habilitated processors, so for software billing purposes, the STSI instruction would report 713.
The MCI has the Nxx format and follows these rules.

- **N** = the capacity setting of the engine:
  - 7xx = 100%
  - 6xx ~ 64.00%
  - 5xx ~ 51.20%
  - 4xx ~ 20.00%

- **xx** = the number of PUs characterized as CPUs in the server

After xx exceeds 15, then all CPU engines are full capacity. Here is a list for all possible values:

- 700, 401 to 415, 501 to 515, 601 to 615 and 701 to 780.

  The software model 700 does not have any CPU engines, it may have ICFs and IFLs.

There are 126 (due to: 80 + (3 x 15) + 1) possible combinations of capacity levels and numbers of processors. These offer considerable overlap in absolute capacity, provided different ways. For example, a specific capacity (expressed as MSUs) might be obtained with a single faster processor or with three slower processors. The hardware cost is approximately the same.

The single-processor version is not recommended because a high priority task in a processor loop can have a significant negative impact. In the above example, the three-way CPC might be a better choice.
3.7 Model capacity identifier and MSU/h

Let us have a summary of all possible Model Capacity Identifiers for each z196 model possible:


Figure 3-7 shows for the z196 top capacity CPCs the amount of MSU/h as measured by the Large System Performance Reference (LSPR) project. This metric (MSU/h) is used by many software companies (IBM with WLC plan) to charge for software.
3.8  z196 frames, cages and I/O drawers (I)

Let us see the big chunks that the z196 is made of, as shown in Figure 3-8. It looks very much like the z10. The major differences are the I/O drawers and the water cooling units, an option for the z196.

z196 models have two frames:

- The A-Frame (at right) is divided into two cages: a PU cage (above) with the processor units (up to 96), central storage (up to 3.0 TB), two I/O cages for I/O channels, plus power and cooling units (MRU), and an optional internal battery feature (IBF). Optionally the I/O cage may be replaced by two I/O drawers.

- The Z-Frame is divided into one I/O cage and two ThinkPads that are used as Support Elements (SEs); the processor controlling all the z196 CPC (and the zBX). The SE is also used as an interface with IBM custom engineers. Z-Frame also contains basic power supplies and an optional IBF. Optionally each I/O cage may be replaced by two I/O drawers.

The I/O cages (and I/O drawers) have I/O cards (located in I/O slots) containing I/O processors known as I/O channels. In total you may have up to 1024 channels, but up to 256 serving a z/OS image in a logical partition.
3.9 z196 frames, cages and I/O drawers (II)

Books
The processor unit (PU) cage (also called the CEC cage) can have up to four components known as books connected via a point-to-point SMP network. These components are called books because they look like books on a library shelf. Each book consists of up to 20 or 24 processor units (PUs), memory (up to 750 GB) and I/O fanout cards. The fanout cards are processor paths to I/O channels in the I/O cages (and I/O drawers). The configuration models vary with the number of books and the number of PUs per book.

Optional battery feature (IBF)
IBF keeps the CPC powered up for up to 10 minutes when there is a power outage (in all four AC feeds). This time is dependent on the number of I/O cages. In practical terms, the IBF can be used as follows:

- To keep the storage contents of the LP running the non-volatile Coupling Facility (CF), then allowing structures to rebuild in the other CF.
- For orderly shutdown of z/OS in case of a longer outage, if the I/O storage configuration has an alternate source of power (which is usually the case).
- To avoid disruption while waiting for a short power outage to pass.
I/O cages
The z196 contains an I/O subsystem infrastructure that uses an I/O cage that provides 28 I/O slots (where channels and adapters are located) and the ability to have one to three I/O cages, delivering a total of 84 I/O slots.

z196 continues to use the Cargo cage for its I/O (as used by the z9 CPC), supporting up to:
- 240 ESCON channels
- 288 FICON Express-8 channels

I/O drawers
I/O drawers provide increased I/O granularity and capacity flexibility and can be concurrently added and removed in the field (FRU), an advantage over I/O cages, which also eases preplanning.

The z196 server can have up to six I/O drawers, four in the A frame and two in the Z frame. There are possible combinations: three I/O cages and six I/O drawers. The third I/O cage requires a Request Price Quotation (RPQ), that means a special manufacturing order.
3.10 NUMA topology

Here we are explaining the need for several books, using the cake analogy, in the z196 design. In a symmetrical multiprocessing (SMP) design, as implemented in z196 where several PUs (up to 96) access (share) the same central storage, there is a limitation for the numbers of such CPUs, performance-wise. This limitation is much smaller than 96.

To overcome such a problem, Non-Uniform Memory Access (NUMA) topology in z196 CPC is introduced. Each set of PUs (no more than 24) has local access to one piece (25%) of central storage (the one allocated in its book) and remote access to other pieces (75%). In other words, the central storage access is non-uniform.

LPAR logic and the z/OS dispatcher together try to guarantee that the majority (more than 80%) of the memory access by one PU is in the memory located in its same book.

As an analogy, consider a baker who decides spontaneously to bake a cake, but only has flour, sugar, and eggs (80% of the ingredients) at home, because it is uneconomical to keep all the ingredients on hand. So the baker needs to go (remotely) to a supermarket for the remaining 20% of the necessary ingredients.
3.11 z196 books

A book has processor units (PUs), central storage, and fanout cards. Books are located in the PU cage in Frame A; see Figure 3-8 on page 179. A book has the following components:

- Multichip module (MCM). Each MCM includes six quad-core processor unit (PU) chips and two Storage Control (SC) chips. Refer to “Multichip module (MCM)” on page 188 for more information about this topic.

- Memory implements Redundant Array of Independent Memory (RAIM), like disk RAID. This technology adds significant error detection and correction capabilities. Memory is formatted in dual in-line memory module DIMM (technology SuperNova x81) with 10 to 30 (14 + 16) DIMMs per book. RAIM parity bits take 20% of total memory. There is no non-RAIM option. So we have 40 to 960 GB RAIM total memory per book and 32 to 768 GB of memory for customer use (20% less).

- Fanout cards, connecting the book with the I/O cages (and in certain cases connecting this book with another book in another CPC), where the I/O channels are. Through the fanout cards:
  - A channel can access data (read or write) in central storage located in a book, moving it to or from I/O controllers along I/O operations.
  - SAPs can talk to channels during the start of the I/O operation or I/O interrupts.

Refer to “Fanout cards in a book” on page 186.
Distributed converter assemblies (DCAs). Three DCAs provide power to the book. Loss of a DCA leaves enough book power to satisfy the books’ power requirements. The DCAs can be concurrently maintained.

Functional service processor (FSP). The FSP card is based on the IBM Power PC microprocessor. It connects to an internal Ethernet LAN to communicate with the support elements in this book and in other books.

About book replacement
With enhanced book availability and flexible memory options, a single book in a multibook system can be concurrently removed and reinstalled for an upgrade or repair; it is a field replacement unit (FRU). Any book can be replaced, including book 0, which initially contains the HSA.

However, this requires that you have sufficient resources in the remaining books to avoid impacting the workload. CPs and memory from the book must be relocated before the book can be removed. Not only do additional PUs need to be available on the remaining books to replace the deactivated book, but also sufficient redundant memory must be available if it is required that no degradation of applications is allowed.

You may want to consider using the flexible memory option. Removal of a book also cuts the book connectivity through its fan out host channel adapters (HCA). The impact of the removal of the book is limited by the use of redundant I/O Interconnect.

PR/SM has knowledge of the amount of purchased memory and how it relates to the available physical memory in each of the installed books. PR/SM also has control over all physical memory and therefore is able to make physical memory available to the configuration when a book is non-disruptively added.

PR/SM also controls the reassignment of the content of a specific physical memory array in one book to a memory array in another book. This is known as the Memory Copy/Reassign function. It is used to reallocate the memory content from the memory in a book to another memory location when enhanced book availability is applied, to concurrently remove and reinstall a book in case of an upgrade or repair action.
3.12 Any to any book connectivity

Book topology
As shown in Figure 3-12, in the z196 CPC, books are interconnected by a point-to-point connection topology (star) with high speed communications links via the L4 caches. This allows every book to communicate with every other book. Data transfer never has to go through another book (cache) to address the requested data or control information. Inter-book communication takes place at the L4 cache level. The star topology is designed to get the maximum benefit of the improved processor clock speed.

The Storage Controller (SC) function regulates coherent book-to-book traffic.
3.13 Fanout cards in a book

![A book picture showing the fanout cards](image)

**Fanout cards**
A fanout card is a processor in charge of moving data in a z196 CPC between:
- The book and the I/O cage (or the I/O drawer) or
- Two books in different CPCs. In this case the link is established between an MVS LP and a Coupling Facility LP located in distinct CPCs.

The connectivity to the I/O cages (or I/O drawers) in the z196 is implemented by InfiniBand technology, which provides a data rate of 6 GB/sec.

Each z196 book has a combination of up to 8 InfiniBand fan out cards, also called Host Channel Adapters (HCA). There are three types of HCA:
- Host Channel Adapter2-C (HCA2-C) provides copper connections for InfiniBand I/O interconnect to all I/O, ISC-3, and Crypto Express cards in I/O cages and I/O drawers.
- Host Channel Adapter2-O (HCA2-O) provides optical connections for 12x InfiniBand for coupling links (PSIFB). The HCA2-O provides a point-to-point connection over a distance of up to 150 m (492 feet), using four 12x MPO fiber connectors and OM3 fiber optic cables (50/125 inches).

z196 to z196 or System z10 connections use 12-lane InfiniBand Double Data Rate (12 x IB-DDR) link at 6 GBps. If the connection is from z196 to a System z9, 12-lane InfiniBand Single Data Rate (12 x IB-SDR) at 3 GBps is used.
The HCA2-O LR fanout provides optical connections for 1x InfiniBand and supports PSIFB Long Reach (PSIFB LR) coupling links for distances of up to 10 km and up to 100 km when repeated through a System z qualified DWDM. This fanout is supported on z196 and System z10 only.

Each fanout has two ports to connect either a copper cable IFB (internal) or optical. There are up to eight HCA2-C fanout cards per book. Then, the maximum number of fanout cards per CPC is 24 with 2 ports each. Multiplying 24 x 2 x 6 MB/sec, we have a 288 GB/sec of total z196 aggregate I/O data rate as referred to in “z196 numeric comparison” on page 172.

No more Memory Bus Adapter (MBA) fanout cards for ICB-4 CF links can be installed in a z196, as in a z10 CPC.
3.14 Multichip module (MCM)

Multichip module (MCM)

It is time to cover the key book component, the MCM. The z196 book has a new Multichip Module (compared with z10) with six new IBM z196 processor chips. Depending on the MCM version (20 PU or 24 PU), we may have from 20 to 96 PUs in the CPC. This new MCM provides a significant increase in system scalability and an additional opportunity for server consolidation. All books are interconnected with very high-speed internal communications links in a fully connected star topology via the L4 cache, which allows the CPC to be operated and controlled by the PR/SM facility as a symmetrical, memory-coherent multiprocessor.

There is just one MCM per book. It contains eight chips and measures approximately 96 x 96 millimeters (same z10 size). MCM contains 103 glass ceramic layers (same as z196) providing interconnection between the chips and off-module environment. It has a cycle time of 0.19 nanoseconds, that is, 5.2 GHz for frequency. Each MCM has:

- Six PU chips, each PU chip with up to 4 PUs. Two MCM options are offered: with 20 PUs (4 chips with 3 PUs and two chips (PU3 and PU5) with 4 PUs), or 24 PUs (all 6 chips with 4 PUs).
- Two SC chips containing a processor called Storage Controller and the L4 cache. Each SC chip is connected to all six PU chips.

L4 cache

L4 cache is shared by all PUs of the book. It has 192 MB. It is managed by Storage Controller and has a store-in cache design. More on cache design in “Storage layers” on page 208.
3.15 Frequency (GHz) in a z196 PU

Here, we discover some technology numbers of the z196 (CMOS 12S).

Frequency is the inverse of cycle time. It is not the only metric when measuring PU performance (speed); which is not linear with frequency. Instead, you need to use Large Systems Performance Reference (LSPR) measurements along with System z capacity planning tools to achieve real client workload CPU sizing.

The PU speed for executing a program depends on:

- Low cycle time and consequently high GHz (frequency) - as provided by the technology (type of circuitry).
- Low number of cycles per average executed instruction (CPAI) - as provided by the design through the use of techniques as pipeline and cache.
- Fewer instructions in the referred program - as provided by the architecture through a powerful set of instructions.

The technology is reaching the physics limits and in the future, we should not expect a growth in frequency as experienced in the past. For example from a z9 EC to a z10 EC the frequency growth was 2.59 times. From z10 EC to z196 the growth is only 1.18 times. Then, the processor manufactures must use new concepts in design and architecture to compensate for the future poor growth in the technology (frequency).
z196 focus
The z196 focus is on balanced system design and architecture to leverage technology, as listed here:

- Low-latency pipeline (refer to “Pipeline in z10 EC” on page 129 for more information) and better cache implementation.
- Optimum set of instructions.
- Virtualization technology allows consistent performance at high utilization, which makes CPU power-efficiency a much smaller part of the system and data center power consumption picture.
- Environmentally sensitive energy efficiency.

However, frequency (GHz) is still significant and important because it is a way for technology to speed up the processors.
3.16 Quad core PU chip

Now, we are zooming the MCM going to the PU chip. All chips in the z196 use Complementary Metal Oxide of Silicon (CMOS) 12S technology, the state-of-the-art microprocessor technology based on ten-layer Copper Interconnections and Silicon-On Insulator technologies.

The new Enterprise Quad Core z196 PU chip is comprised of three or four PUs. A core is another word for PU. It contains 3.5 km of wire, and has 1.5 billion transistors in a 450-mm² area. A schematic representation of the PU chip is shown in Figure 3-16.

**Processor Unit (PU) or core**

The four PUs (cores) are shown in the corners of Figure 3-16. They include the private L1 and L2 caches, plus all microprocessor functions. COP indicates the two coprocessors, each of which is shared by two of the four cores. The coprocessors are accelerators both for data compression (Zivv Lampel algorithm) and cryptographic functions. Do not confuse this coprocessor with the Crypto Express-3 that is located in the I/O cage.

As mentioned, there are six PU chips on each MCM. The six PU chips come in two versions. For models M15, M32, M49, and M66, the PUs on the MCM in each book are implemented with a mix of three active cores and four active cores per chip (4 x 3 cores active, plus 2 x 4 cores active), resulting in 20 active cores per MCM. All MCM in these models have 20 active cores. This means that a Model M15 has 20 PUs, a Model M32 has 40 PUs, a Model M49 has...
60 PUs, and a Model E66 has 80 PUs. The model M80 has the 6 PU chips with 4 cores each, then 96 active PUs.

MCM and spare PUs
CPC-wide, two spare PUs (cores) are available that may be allocated on any MCM in the system.

L1 and L2 cache
Each PU has two private caches:
- A first level 192 KB on chip Level 1 cache (L1) that is split into a 64 KB L1 cache for instructions (I-cache) and a 128 KB L1 cache for data (D-cache).
- A second level on chip Level 2 cache. It has a size of 1.5 MB per PU. The two levels on chip cache structure are needed to optimize performance so that it is tuned to the high frequency properties of each of the PUs.
- Both caches have a store-through design. Refer to “Storage layers” on page 208, to learn more about store-through design.

L3 cache
The L3 cache with 24 MB in the PU chip is shared by all four PUs in the PU chip. It has a store-in design. Refer to “Storage layers” on page 208, to learn more about store-in design.

Other PU chip elements
GX indicates the I/O bus controller that controls the interface to the Host Channel Adapters (HCA) accessing the I/O cage.

The Memory Controller (MC) is the gate to access the central storage banks located in this book (DIMM). Refer to “MCUs and DIMMs in a z196 book” on page 221.

As you can imagine, Gx and MC adapters control intensive traffic between the PUs, memory, I/O and the L4 cache located on the SC chips.
3.17 PU chip coprocessor

Figure 3-17  PU Chip coprocessor

**PU chip coprocessors (accelerators)**
There are two coprocessors in the PU chip, each one shared between two PUs. It offloads hardware circuit functions from the PU itself. When the coprocessor is executing the PU request (or the PU request is in a queue), the requesting PU is in microcode loop. The coprocessor has two main parts implementing two functions: data compression and cryptography.

**Data compression engine**
Activated by the instruction Compress in the PU. It executes static dictionary compression and expansion. The dictionary size is up to 64 KB (8K entries). It has local 16 KB cache per core for dictionary data.

**CP assist for cryptographic function (CPACF)**
Activated by several instances of the Cypher instruction when executed in the PU. It is made of Crypto Cypher and Crypto Hash components. It executes Clear Key crypto algorithms. Processor unit (PU) core details.

In the z196 using the z/OS System Control Program, CEX3 cryptographic hardware can only be used through ICSF. The CPACF hardware can be accessed either via ICSF Application Programming Interfaces (APIs) or by native instructions. ICSF is a standard component of z/OS that provides the APIs by which applications request cryptographic services. Thus ICSF relieves the application from dealing with the complexity of the cryptographic hardware.
communication. However, these ICSF services are operating software path lengths which have to be added (from an application's point of view) to the execution time of the cryptographic hardware. As mentioned in the description of the CPACF cryptographic hardware, an application program can use this hardware by invoking the MSA machine instructions. However, there is also an API provided by ICSF.

Prior to System z10 EC GA3 all CPACF functions required the use of clear keys. With z10 EC GA3 and beyond the CPACF MSA architecture interface was extended to support the use of IBM zEnterprise 196 Cryptographic performance.

CPACF protected keys are wrapped with a CPACF wrapping key and are never in operating system addressable memory in an unwrapped state. Using CPACF functions with protected keys leverages the performance benefits of CPACF hardware while providing added key protection required by security sensitive applications. This section presents CPACF encryption rates using the MSA architecture instructions for both clear key and protected key modes of operation.

The results show that protected key operations have lower encryption rates than the equivalent clear key operation. This is expected because the protected key needs to first be unwrapped within the CPACF (using a CPACF wrapping key) before the requested instruction can be processed. As the data length increases, the key manipulation is a less dominant factor and the protected key rate approaches the clear key rate.

All test cases are written in IBM System z® Assembler Language issuing the System z Message Security Assist (MSA) Architecture cryptographic operation instructions as indicated with each group.
Processor unit (core) details
Our zoom reaches the end, the PU itself. Figure 3-18 shows a z196 PU (core).

The z196 PU instruction set has 984 instructions. Due to performance reasons 762 instructions are hardwired and not microcoded. 246 of the most complex z/Architecture instructions are implemented via millicode. Some instructions have both hardware and millicode implementation. Millicode is PU internal code at a higher level than microcode and is very similar to Assembler. Millicode is required to implement the more complex instructions of the instruction set, such as Start Interpretive Execution (SIE).

Each PU is a superscalar (several instructions being executed at the same time), implementing a pipeline with out-of-order execution. Refer to “Pipeline concept within a PU” on page 200 and “Out of order execution” on page 202 for more details about such concepts.

Some details of a z196 PU:
- Six execution units. 2 fixed point (integer), 2 load/store, 1 binary floating point, 1 decimal floating point.
- Up to three instructions decoded per cycle (versus two in z10).
- Up to five instructions/operations executed per cycle (versus three in z10).
- Instruction execution can occur out of (program) order.
- Memory address generation and memory accesses can occur out of (program) order.
- Each core has special circuitry to make execution and memory accesses appear in order to software.
There are 246 complex instructions executed by millicode and another 211 complex instructions cracked into multiple operations.

Each core has three private caches:
- 64 KB 1st level cache for instructions, 128 KB 1st level cache of data
- 1.5 MB L2 cache containing both instructions and data

The acronyms presented in Figure 3-18 are as follows:

- **IFB** - Instruction fetch and branch (I Buffers)
- **ICM** - Instruction cache and merge (64 K I Cache)
- **IDU** - Instruction Decode
- **ISU** - Issue (Out of order (OOO) execution)
- **XU** - TLB, DAT (Address Translation, I/O Interrupt control). Translation Look-aside Buffer (TLB) has 512 entries. These entries are used by dynamic address translation (DAT) to keep the real addresses of the 512 most-referenced pages.
- **LSU** - Load/Store (128 K D Cache)
- **FXU** - Fixed Point (Execution)
- **BFU** - Binary/ floating point (Execution)
- **DU (DFU)** - Decimal and Floating point Operations (Execution). Each PU has a Hardware Decimal Floating Point unit (HDFU) to accelerate decimal floating point transactions. It conforms with the standard IEEE 745R. This is expected to be particularly useful for the calculations involved in many commercial transactions (previously they were performed by software routines).
- **RU** - Checkpoint and retry controls
3.18 Storage controller (SC) chip in MCM

This topic explains the role of the Storage Controller processor and its L4 cache in the MCM.

On each book there is one MCM that contains up to 24 PUs. The MCM has six PU chips and two SC chips. Each PU chip has up to four cores, which can be characterized as CPs, IFLs, ICFs, zIIPs, zAAPs, or SAPs. Two MCM capacity models are offered, 20 or 24 PUs.

There are two SC chips on the MCM. Each SC chip contains 1.6 billion transistors and measures 24.4mm x 19.6mm. An SC chip has:

- 48 MB of L4 cache, resulting in a combined L4 cache size of 96 MB (2 x 48 MB) per SC chip and 192 MB per book.
- A Storage Controller processor (L4 Controller in Figure 3-19), which controls the communication:
  - Between the L2/L1 cache in the PUs, the L3 cache in the chip and the L4 cache in the same MCM. This is done by several bidirectional data buses.
  - Of the cross-point switch for L4-to-L4 traffic to up to three remote books by three bidirectional 16-byte data buses.
Storage controller function
The Storage Controller (SC) acts as a coherency manager. It is responsible for coherent traffic between the L4 caches in a multi-book system, and between the L4 cache and the chip local L3 and private PU L2/L1. Note that the L4 cache is shared between all PUs of a book.

The SC has directory information for all Lx caches in the book. If there is a change in the contents, the SC can invalidate the old copy in other Lx caches. The SC processor in an SC chip also controls (through MCU) the read and write of data in between the central storage (also called L5) and the L4 cache located at the SC chip.

The SC processor also acts as an L4 cache cross-point switch for L4-to-L4 traffic to up to three remote MCMs or books by three bidirectional 16-byte data buses with a 3:1 bus/clock ratio. Figure 3-19 on page 197 displays the various elements of the PU chip. In the figure, most of the space is taken by the DRAM L4 cache. L4 Controller indicates the SC function of the chip for point-to-point inter book communication. Directory and addressing function locations are also shown.
3.19 z196 book recapping

Recapping the z196 book
Figure 3-20 displays the major z196 book components (the two frames are not pictured here).

The book, which is a component of the PU cage, is shown in the top right corner.

There are four books. One MCM is shown, with six PU chips and two SC chips.

Each PU chip has up to four PUs, each with L1 and L2 caches, one L3 cache and two coprocessors for data compressing and crypto hardware functions.

Each SC chip has an L4 cache and an SC processor.
3.20 Pipeline concept within a PU

Pipeline concept within a PU

To understand with more detail how a PU works when executing a program, we need to cover the pipeline concept.

To build a car, serial tasks (steps) need to be executed. Similarly, when executing program instructions, sequential tasks also need to be executed. The term “pipeline” implies executing, in parallel, different tasks for different cars (or instructions). The main objective of a pipeline is to increase throughput (that is, the number of executed instructions per time unit), and not to decrease the average time to execute one instruction. In other words, the pipeline aims to decrease the average number of cycles per executed instruction (CPAI) due to parallelism.

Within the z196 PU there are a few special micro processors, each one executing a very specific step along the execution of a program (a set of instructions):

- Get the instruction virtual address in the PSW.
- Translate this virtual address to a real address through the DAT.
- Fetch the instruction from cache (L1 or L2 or L3 or L4) or from central storage.
- Decode the instruction; that is, if the instruction is microcoded, find the microcode address (in control storage) associated with its execution.
- If there is an input storage operand, calculate its virtual address through the contents of the base register plus the displacement.
- Translate this virtual address to a real address through the DAT.
- Fetch the operand from cache (L1 or L2 or L3 or L4) or from central storage.
- Execute the instruction.
- If there is an output storage operand, derive its virtual address through the contents of the base register plus the displacement.
- Translate this virtual address to a real address through the DAT.
- Store the output operand in cache (L1).
- Depending on the instruction, set the condition code in the PSW.

**Techniques for improving an instruction pipeline**

In z196 211 complex instructions can be cracked into multiple internal operations. Techniques exist for speeding up an instruction pipeline:

- Execute more than one instruction in the same cycle (superscalar). This is implemented by adding resources onto the PU to achieve more parallelism by creating multiple pipelines, each working on their own set of instructions. A **superscalar** server is based on a multi-issue architecture. In such a PU, where multiple instructions can be executed at each cycle, a higher level of complexity is reached because an operation in one pipeline may depend on data in another pipeline. A superscalar design therefore demands careful consideration of which instruction sequences can successfully operate in a multi-pipeline environment. The z196 PU is superscalar, which means that it is able to execute up to five instructions per cycle (versus 3 in z10) and up to three instructions decoded per cycle (vs. 2 in z10). However, these concurrent executed and concurrent decoded instructions are not any set of five instructions.
3.21 Out of order execution

Perform out-of-order (OOO) execution
This implies that the sequence of instructions presented in a program is not the exact sequence, where the instructions are executed. For example, in Figure 3-22, the execution sequence is 1, 5, 4, 2 and 3. In the same example, 5 is executed before 2, 3 and 4 because it is ready first (no need to access memory), and its inputs do not depend on the output of 2, 3 and 4. Instructions 2 and 4 are being delayed by a storage operand fetch. The previous z10 PU does not implement such function; rather, it performs in-order execution.

Perform out-of-order fetch
Instructions having memory operands may suffer multi-cycle delays to get the memory contents, if such contents are not at cache levels. To overcome these delays, the PU continues to fetch (single cycle) instructions (and its data) that do not cause delays. The technique used is called out-of-order operand fetching.

This means that some instructions in the instruction stream are already under way, while earlier instructions in the instruction stream that cause delays due to storage references take longer. Eventually, the delayed instructions catch up with the already-fetched instructions and all are executed in the designated order. The z196 and z10 PUs implement such function.
3.22 z196 instructions

- z196 has 984 instructions of which 762 are hardware implemented
- 110 new instructions for Java, WebSphere, multiple arithmetic formats and Linux workloads
- New instructions categories:
  - High-word facility and interlocked-access facility
  - Load/Store on condition facility
  - Distinct operands facility
  - Integer to/from floating point conversions
  - Decimal floating point quantum exceptions
  - New crypto functions and modes
  - Virtual architectural level
  - Non-quiescing SSKe

**Figure 3-23  z196 instruction set**

**z196 instruction set**

Here we present a highlight view of the enhancements in the z196 instruction set compared with z10. The 196 PU instruction set has 984 instructions as described in z/Architecture Principles of Operations, SA22-7832. This makes z196 a real Complex Instruction Set of Computing (CISC) machine.

Compared with z10 there are 110 new instructions designed to improve Java, WebSphere®, multiple arithmetic formats, and Linux workloads. Recompiled application code produces further performance gains through these new instructions.

The new instruction categories follow:

- New truncating and OR inexactness Binary Floating Point rounding mode.
- New Decimal Floating Point quantum exception. Eliminates need for test data group for every operation.
- The virtual architecture level allows the z/VM Live Guest Relocation Facility to make a z196 behave architecturally like a z10 system. It facilitates moving work transparently between z196 and z10 systems for backup and capacity reasons.
- On Non-quiesce SSKE:
  - Significant performance improvement for systems with large number of processors (typically 30+).

High-Word Facility (30 new instructions)
- Independent addressing to high word of 64-bit General Purpose Registers (GPR).
- Effectively provides compiler software with 16 additional 32-bit registers.

Interlocked-Access Facility (12 new instructions)
- Interlocked (atomic) load, value update and store operation in a single instruction
- Immediate exploitation by Java

Load/Store-on-Condition Facility (six new instructions)
- Load or store conditionally executed based on condition code
- Dramatic improvement in certain codes with highly unpredictable branches

Distinct-Operands Facility (22 new instructions)
- Independent specification of result register (different than either source register).
- Reduces register value copying.

Population-Count Facility (one new instruction)
- Hardware implementation of bit counting roughly five times faster than prior software implementations

Integer to/from floating point converts (21 new instructions)
3.23 Non-quiesce SSKE instruction

- Non-quiesce SSKE enhances SMP scaling for z/OS images to up to 10% performance.
- SSKE instruction alters all or some of the 4K-byte frame storage key bits.
- A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.
- In a z196, it is implemented the non-quiesce SSKE. This function avoids the serialization and checkpoint-synchronization function.

**Figure 3-24 Non-quiesce SSKE Instruction**

**Non-quiesce SSKE instruction**

Figure 3-24 is referring to a significant performance improvement in a z196 PU, mainly for CPCs with large numbers of processors (typically 30+). It enhances SMP scaling for z/OS images up to a 10% performance increase. Exploited by z/OS 1.10 and above (with PTFs).

The improvement is called non-quiesce SET STORAGE KEY EXTENDED (SSKE) instruction.

**Quiesce SSKE instruction**

The SSKE instruction, when issued by the z/OS real-storage manager component, alters all or some of the 4K-byte frame storage key bits. This happens every time a page is allocated in a frame. The value of the first five storage key bits (called protection bits) is set according to the page subpool number that was gotten at page Getmain.

Although not directly related to dynamic address translation (DAT), the storage key of a 4K-byte block is maintained in the processor's translation look-aside buffer (TLB). Due to the TLB page-table copy that is maintained in a multiprocessor configuration, the SSKE instruction requires that the new storage key be broadcast to all processors in the configuration because other processors may have cached copies of the key in their TLBs. This broadcast operation—which may take hundreds of cycles—must complete before the SSKE instruction can complete. As you can see, in a CPC with a large number of PUs the...
performance issue may be significant.

The next two paragraphs are extracted from the Principles of Operation:

“A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed. It means that all the CPUs in the CPC are quiesce because integrity reasons (as concurrent update of the storage key bits) along the SSKE execution.

This serialization and checkpoint-synchronization also guarantees that for any store access, by any CPU or channel program, completed to the designated 4K-byte block either before or after the setting of the key by this instruction, the associated setting of the reference and change bits to one in the storage key for the block also is completed before or after, respectively, the execution of this instruction.”

In some cases—for example, when only a reference or change bit is being altered, but the protection key value remains the same—broadcasting the key change is superfluous, and wastes processor cycles.

In order to improve the performance of such instructions, a two-step implementation was designed in the z/Architecture:

- At z10 CPCs a conditional-SSKE facility is implemented. This facility has two aspects:
  - The SSKE instruction may be used to set all or portions of the storage key based on program-specified criteria.
  - Certain functions of the key-setting operation may be bypassed and the serialization and checkpoint-synchronization function may not be performed when the update is for change and reference bits. It may cause 10% of overall processor performance improvement.

- At z196 the non-quiesce SSKE is implemented. This function minimizes the serialization and checkpoint-synchronization function even when the update applies to the protection bits.
3.24 z10 EC and z196 cache design comparison

Here we pinpoint the major differences and similarities between the cache design in z10 EC and z196.

They are as follows:
- z10 EC has three layers of cache (L1, L1.5, and L2), z196 has four (L1, L2, L3 and L4).
- The majority of the cache sizes are not the same in the two families of CPCs.
- z10 EC has two PU private caches L1 and L1.5; z196 has two PU private caches L1 and L2.
- In a sense L1.5 in z10 EC is renamed as L2 in z196.
- z10 EC has no PU chip shared cache; z196 has one PU chip shared cache, the L3.
- In a sense L2 in z10 EC is renamed as L4 in z196.

The line (unit of data transport between caches and memory) has the same size in both (z10 EC and z196) design, that is, 256 bytes.
3.25 Storage layers

Before we really start the explanation of the z196 cache design, we take a look at a very basic idea in data processing, that is, the storage layers (or storage pyramid).

Figure 3-26 shows layers of storage to keep data and programs in a z196 computing system.

As you move up in the layers:
- Price per byte goes up (faster technology).
- The time to access data goes down (better performance).
- The size goes down.

Somewhere, there is a line separating the storages that are volatile (above) from the ones that are nonvolatile. A volatile memory needs a constant flow of energy to keep the data information. Taking in consideration that the z196 Internal Battery Feature is just an interim feature, we may say that the above mentioned line separates the DASD and TAPE caches from the Coupling Facility structures.

We may say that hardware and software (operating system or not) should try to keep the most important data (and programs), normally the ones most frequently accessed in the top layers.

For example, a processor program with its data and instruction in a top layer cache (as L1 or L2), needs much fewer cycles to execute the same logic. In other words, its average number of cycles per instruction (CPAI) is less than when it needs to frequently go to central storage.
The performance of a layer (z196 four layers of cache included) depends on:

- Size, the larger the worst, because more time to search for specific contents
- Distance from the processor, the longer the worst
- Technology
- Algorithm to keep the most referenced data at upper layers
- Population of tasks or programs fighting for hits in re-visits. The higher the population, the less chance of hits in re-visits.

**Store-through concept**

A storage layer is store-through; when any update is made to it, it must be synchronously propagated to the lower layers. The L1 and L2 caches are examples of such. The reason for these caches of being store-through is to make the information about the update available to other PUs located in other PU chips or even in other books. Then the store-through concept exists for integrity reasons.

**Store-in concept**

A storage layer is store-in; when any update is on it, it is delayed and asynchronously propagated to the lower layers. The L3, L4 and DB2 central storage buffers are examples of such. The reason for L4 caches being store-in (the update of central storage is delayed) is because all L4 caches (through the SC) can be accessed by all PUs in all chips and in all books. Being store-in you may improve performance (by delaying the update to lower and slower storage layers) without causing any integrity issues.
3.26 z196 cache design (I)

Figure 3-27 z196 cache design

Four levels of cache

Caches are fast memories used to avoid the need to access lower-level memories (in the storage pyramid) that are cheaper, larger and slower. We may say that the use of caches by PUs executing programs decreases the average number of cycles needed to execute such programs. In the z196, there are four levels of cache (L1, L2, L3, and L4) to improve performance by firstly reducing access to real storage (sometimes called L5) and secondly reducing access to a lower-level cache.

Caches in z196 CPC are formatted in lines of 256 bytes and are not directly visible to programs, although a new prefetch data instruction is provided to allow a program to prefetch or release a line. This way a program can manage cache to some degree in order to improve cache-hit ratios (not recommended for normal programs).

Due to performance issues, cache sizes are being limited by ever-decreasing cycle times because they must respond quickly without creating bottlenecks. Access to large and distant caches costs many more cycles. This phenomenon of shrinking cache sizes can be seen in the design of the z196 compared with the z10 EC in order to accommodate decreased cycle times. The L2 cache in z196 has 1.5 MB and the equivalent L1.5 cache in 10 EC has 3 MB.

The distance to remote caches is also a consideration; for example, the L4 cache is not located in the PU chip and might be in another book. To minimize the access to L4, the z196 introduces the L3 cache shared by all PUs of the same PU chip. And to minimize the access to the L3 cache, the PU private L2 cache has been introduced as well. These
intermediate-level caches reduce traffic to and from the L4 cache. With the L2 and L3 caches, requested lines are sent to the L4 cache only when there is a line cache miss in the upper level (layers) caches.
3.27 z196 cache design (II)

**L1 cache**
Each PU has its own private 192 KB Cache Level 1 (L1), split into 128 KB for data (D-cache) and 64 KB for instructions (I-cache). The reason for splitting data and instructions is because the pattern and nature of the reference is different; for example, data is often changed, but instructions are very seldom changed. The PU itself only fetches and stores instructions or operands (data) from or to the L1 cache. The L1 cache is designed as a store-through cache, meaning that altered data is immediately (synchronously) stored to the next level (L2). All the lines that are located in the L1 cache are also located in the corresponding L2 cache.

**L2 cache**
Going down in the pyramid, the next level of memory is the L2 cache. Located side by side with a PU, it is a private cache as well. It has a size of 1.5 MB mixing data and instructions. It is also a store-through cache to L3. All the lines that are located in the L2 cache are also located in the corresponding L3 cache.

**L3 cache**
Going even further down in the pyramid, the next level of memory is the L3 cache. Located in the PU chip it is shared by all the PUs from this chip. It has a size of 24 MB mixing data and instructions. It is a store-in type of cache. The update line is copied to L4 on demand or when the L3 cache is reaching a threshold.
**L4 cache**

The next down layer is the L4 cache. Located in two SC chips (96 MB per chip) in the MCM, it is shared by all the PUs from this book. L4 is accessed by all SCs in the z196 CPC, even the ones located in other books. It has a size of 192 MB mixing data and instructions. It is a store-in cache to central storage. It contains all the lines (and more) in the six L3 caches in its book.

Each L4 cache has a direct path to each of the other L4 caches in remote books through point-to-point (any-to-any) connections. SC processor services provide the communication between L4 caches across books. The L4 cache contains the most referenced lines by the PUs of its book and not only the lines coming from the central storage located in its book.

The de-stage to central storage is performed when L4 occupancy reaches some threshold. The performance advantage of being store-in is that the PU does not need to wait for a slow store in memory. Also, if the same line is updated several times when in an L4 cache, the store to central storage is done just once.

The z196 uses a least-recently used (LRU) algorithm to decide the correct population of lines in each cache layer. When L1, L2, and L3 layers are full, the least referenced lines are purged. When L4 is full, the least referenced and unchanged lines are purged and the least referenced and changed are copied to the right central storage DIMM in a specific book.

**Repeating the same program in the same PU**

A very important concept is that, when a program executing on behalf of a TCB/SRB is running in one PU, its lines (with code and data) are naturally loaded in the private L1/L2 caches, in the L3 PU chip cache and in the L4 book. This load may cause the exclusion of lines belonging to programs that ran previously in such a PU. However, if for any reason the same program is back in the same PU, chances are that their residual lines are still located at least in the larger caches. As a consequence, repeating the same program in the same PU increases the program speed, consistently causing less processor time being executed.

Whenever an operand or instruction needs to be fetched by the PU, the L1 cache is inspected first. When the operand or instruction is found in the cache, it is known as a “hit”. When an operand or instruction is not found, it is known as a “miss”. In the case of a miss (less than 10%), the search sequence is private L2, PU chip shared L3, book L4, other book L4 and then central storage.

**Changing the contents of the L1 cache**

When a PU executes an instruction that changes the contents of the output operand in its L1 cache, the following actions are executed:

- Posting the other PUs, which may have in their L1 or L2 cache an out-of-date copy of the line changed by the first PU.

In the z196, the SC located in each MCM book has the L1/L2 cache directories of all PUs of its book. When a PU alters the contents of one of its L1 line, the SC is directed to invalidate this element in the L1/L2 cache directory of other PUs in the same book, but without disturbing those PUs. This update information is passed to the other SC (in other books) to execute the same invalidation, if needed.

However, just switching on the bit does not guarantee that all PUs will see the most recently updated copy at the same time. To guarantee the full coherency of memory, the program must execute serializing instructions, as described in z/Architecture Principles of Operations, SA22-7832.
With the execution of such instructions (such as the Compare and Swap instruction), all previous storage accesses are completed, as observed by other PUs and channels, before the subsequent storage accesses occur.

- Not synchronously with the update, define a place where the other PUs can fetch the most updated copy of such contents, if they need to do so. In a z196, the updated contents are copied synchronously (store-through) from the L1 cache to the L2 cache, and from the L2 cache to the L3 cache, and sometimes from the L3 cache to the L4 cache, which is a book’s “global memory” where other book PUs (through their SC) can reach the most updated copy of the line.
3.28 HiperDispatch concepts

Software and hardware cache optimization
To achieve a good PU performance, the z196 design has the following targets:

- A very high percentage of instructions and data must be fetched from the L1/L2 caches (hits). A target of more than 90% is desirable.
- In the case of an L1/L2 miss, it is expected that more than 90% of such (previous) misses will be honored from the local (same PU chip) L3 cache.
- In the case of an L3 miss, it is expected that more than 90% of such (previous) misses will be honored from the local (same book) L4 cache.
- In the case of a local L4 miss, it is expected that more than 90% of such (previous) misses will be honored from remote (other book) L4 caches.
- In the case of a remote L4 miss, it is expected that more than 90% of such (previous) misses will be honored from local (same book) central storage.

As you can see, only in a very small percentage of fetches, may the instruction or operand be fetched from the central storage of other books (the worst performance scenario). To achieve such design goals, several functions are implemented:

- Each SC tries to keep the data it fetched (based on this book’s PU requests) from the central storage or L4 of another book in its L4 cache book. In this way, the next fetch request to the same data will be L4 local.
- Each LPAR tries to dispatch the same logical processor in the same physical processor.

Figure 3-29  HiperDispatch example

- z/OS parm for HIPERDISPATCH=NO
- Two levels of Dispatch
  - z/OS Tasks
  - PR/SM Logical Processors
- Due to better cache alignment, customer obtains improvement in system capacity
- Zero customer application changes required (WLM goal tuning may be needed)
The z/OS dispatcher will try to dispatch the same TCB/SRB in the same logical processor, or at least in the same group of logical processors.

**Introducing HiperDispatch**

The last two functions are called HiperDispatch. HiperDispatch assures that, as much as possible, the same TCB/SRB is dispatched in the same logical processor and the same logical processor is dispatched in the same physical processor, as pictured in Figure 3-29 on page 215. This smart dispatching tries to decrease the number of addresses referenced by a physical processor. This directly improves the use of cache, and consequently the use of Translation Look Aside Buffers (TLB) and Access Look Aside Buffers (ALB). This is the reason that we are studying HiperDispatch after the explanation about z196 cache design.

Then, in the z196 with HiperDispatch, we are one step further: PR/SM (LPAR) and z/OS now work in tandem to more efficiently use processor resources.

Figure 3-29 on page 215 shows, on the left side, that with HIPERDISPATCH=NO, the physical processors run different tasks from different logical partitions. On the right side of the figure, with the option YES, the physical processor 1 only switches between task A and task C from the same logical partition.
3.29 The clerk dilemma

Let us have an analogy to introduce the concept of HiperDispatch.

Figure 3-30 is showing a bank with three clerks and customers in just one line (as recommended by the queueing theory). You are the first in line. You already know that the lady clerk in the middle will deliver a shorter service time because she knows you and she understands faster what you need. However, the clerk that is now available is the third one. What do you do, is called the clerk dilemma: “Increase your queue time, letting other customers be served by the third clerk, and believing that the decrease in service time will compensate the increase in the queue time? Or you do the opposite?”

This is the dilemma of z/OS in the function called HiperDispatch.

Significant potential for performance improvement exists by redispaching interrupted TCB/SRB programs to the same physical PU, where they ran prior to the interruption, thereby maintaining affinity to private or shared cache contents. What if this PU with affinity is busy and there are other available PUs (in wait state)?

To balance these two factors—keeping TCB/SRB programs on the same PU and at same time minimizing queue delays—HiperDispatch algorithms aim to maintain several z/OS TCB/SRB queues (affinity node) served by several PUs that share an L4 cache, or in other words, they belong to the same book.
Thus, the extra delay to move a line from one L4 to another L4 is avoided. Instrumentation data was essential in tuning HiperDispatch scheduling algorithms to achieve this balance. Remember that the LPAR is trying to keep the same logical PU running in the same physical PU.

**HiperDispatch logic**

HiperDispatch is available only through the vertical processor management (VCM) architecture hardware component and specialized z/OS V1R9 functions.

This function, as exploited by the z/OS dispatcher, is described in *z/Architecture Principles of Operations*, SA22-7832, under the name Configuration Topology Facility. This facility provides additional topology awareness to the operating system so that certain optimizations can be performed to improve cache hit ratios and thereby improve overall performance. With this function LPAR divides the logical PUs of a logical partition into three groups:

- Vertical high, a sort of dedicated logical PU
- Vertical medium, a logical PU shared
- Vertical low, generally made offline (parked) by WLM because in most of the cases they are defined in excess

HiperDispatch combines the dispatcher actions and the knowledge that PR/SM has about the topology of the PUs in the CPC. For that purpose, the z/OS dispatcher manages multiple TCB/SRB queues with an average number of four logical processors per queue. It uses these queues to assign work to as few logical processors as are needed for a given LPAR workload. So even if the LPAR is defined with a large number of logical processors, HiperDispatch will optimize this number of processors nearest to the required capacity. The optimal number of logical processors to be used is kept within a book boundary wherever possible, thereby preventing L4 cache misses that would have occurred when the dispatcher could dispatch work wherever a processor might be available.
3.30 z/OS dispatcher logic in HiperDispatch

- Logical CPUs (target is 4) assigned a node with consideration for book/chip boundaries based on PR/SM
  - Used by z/OS to assign logical CPUs and work to nodes
- Periodic rebalancing of task assignments to nodes
- Assign work to the minimum number of logical CPUs needed to use weigh
  - Expand use of remaining logical CPUs to use white space
- May require changes to WLM policies for important work
- Single HIPERDISPATCH=YES parameter in IEAOPTxx to dynamically activate HiperDispatch without IPL. IRD management of logical CPUs is turned OFF

Figure 3-31 z/OS dispatcher and HiperDispatch

HiperDispatch considerations
The HiperDispatch function is dynamically activated through HIPERDISPATCH=YES in the IEAOPTxx member in Parmlib. In this case, the Vary Logical CPU Management function of IRD is automatically switched off.

The z/OS dispatcher tries to use a minimum number of active logical processors in an LPAR. This is done to increase the probability of having the same logical processor execute the same dispatchable unit (TCB/SRB). (In order to dispatch a TCP/SRB in a logical processor, the z/OS dispatcher must be running in that logical processor.) The HiperDispatch function engages in a constant communication between the LPAR code and the z/OS dispatcher.

This feature is available with z/OS V1R9, and for z/OS V1R7 with a web deliverable.
3.31 Central Storage design

Central storage, as previously mentioned, is distributed by the four books.

Each basic memory element is the Dual In-line Memory Module (DIMM). The DIMM is a series of random access memory integrated flip-flop circuits (a bit is implemented through a flip-flop). Each DIMM may have a different amount of data: 4, 16 or 32 GB.

Memory in a z196 book is accessed by an MCU. There is one MCU per PU chip. Then, we have six MCUs per book. However, only three of them (PU chip 0, PU chip 1 and PU chip 2) are working at the current implementation; refer to Figure 3-32.

MCU implements a non-associative memory concept; that is, the Storage Controller delivers the z196 book/memory physical address (refer to “Addresses and addresses” on page 223) and the MCU returns the contents of central storage associated with that address for a memory read (256 bytes). The contents are moved to the L4 cache of the local book. Writes from L4 to MCU are done when the L4 cache reaches a threshold of occupancy (L4 cache is store-in). Only the modified lines are copied in the DIMM memory.
3.32 MCUs and DIMMs in a z196 book

Each of the three MCUs access five groups (also called memory channels) of up to two DIMMs each.

The fifth channel (and its DIMMs) in each z196 MCU is required to implement memory as a Redundant Array of Independent Memory (RAIM). This technology adds significant error detection and correction capabilities. Bit, lane, DRAM, DIMM, socket, and complete memory channel failures can be detected and corrected, including many types of multiple failures. Then, RAIM redundancy implementation adds a 5th channel for check bits. It can survive catastrophic DIMM failure. The cost of RAIM is 20% more memory, not charged to customers.

Then the maximum memory size per book is: 32 GB x 30 DIMMs, that is, 960 GB, with 768 MB for customer usage.

Then, the maximum memory size per z196 CPC is: 960 MB x 4, that is, 3840 MB with 3072 MB for customer usage.

Note that memory sizes in each book do not have to be similar; different books may contain different amounts of memory.

Memory can be purchased in increments of 16 GB, up to a total size of 256 GB. From 256 GB, the increment size doubles to 32 GB until 512 GB. From 512 GB to 944 GB, the increment is 48 GB. Beyond that, up to 1520 GB, a 64 GB increment is used.
3.33 Purchase memory offerings

There are several purchased memory options when buying central storage in a z196:

- **Purchase Memory**
- **Hardware System Area**
- **Standard Memory**
- **Flexible Memory**
- **Plan Ahead Memory**

**Purchase memory offerings**

Here we describe the different plans offered by IBM when purchasing memory (central storage):

- Purchase Memory - It is memory available for assignment to logical partitions.
- Hardware System Area (HSA) - Standard 16 GB outside customer memory, for system use
- Standard Memory - Provides minimal physical memory required to hold base purchase memory plus 16 GB HSA. It has the least cost. This plan splits memory as equally as possible among books, using the smallest possible cards for purchased memory.
- Flexible Memory - Provides additional physical memory needed to support the activation base customer memory and HSA on a multiple book z196 with one book out of service. Then, it supports Enhanced Book Availability, splitting memory as equally as possible among books, using large enough cards to ensure purchased memory remains available, if any one book is removed.
- Plan Ahead Memory - provides additional physical memory needed for a concurrent upgrade (LIC CC change only) to a preplanned target customer memory.
3.34 Addresses and addresses

- Virtual address (CP) to a Real Address (DAT)
- Real Address (CP) to a z/Architecture Absolute Address (Prefixing)
- z/Architecture absolute address to z10 EC Absolute Address (LPAR)
- z10 EC Absolute Address (LPAR) to z10 EC Book/memory physical address (Storage Controller)

Addresses and addresses

Figure 3-35 shows that along the execution of a program there is a set of address translations, in order to find the location in memory of the instruction or the operand.

During program execution, the instruction address in PSW is virtual and its operand addresses are also virtual in the base registers plus displacement. To access the instruction and the operands in caches or central storage, these virtual addresses must be translated and converted through several steps into z196 book memory physical addresses. Figure 3-35 shows the multiple stages during that process:

- Virtual address to real address
  Executed by DAT through the use of central storage tables and TLB. The operating system sets up the translate controls (CRs) and translate tables.
- Real address to a z/Architecture absolute address
  Executed by the hardware function Prefixing using the PVR register. The absolute address allows each processor to have a unique 8 KB PSA. Refer to z/Architecture Principles of Operations, SA22-7832, to learn about the concept of prefixing. Prefixing proceeds as follows:
  - If the real address is equal to 0 KB - 8 KB (PSA) - forward prefix
  - If the real address is equal to the Prefix Register - reverse prefix
– If the real address is not equal to the PSA address, or not equal to the prefix register address, then no prefix is used and the real address is an absolute address.

► **z/Architecture absolute address to z196 absolute address**

Executed by the LPAR, as follows:

– LPAR sets up the logical partition relocate offset at image activation time.

– Adds a “relocate” offset to a z/Architecture absolute address. If the LP is the first in central storage, its relocate offset is the size of the HSA. If the logical partition is second in central storage, its relocate address is the sum of HSA size plus the first LP size.

► **z196 absolute address to z196 book memory physical address**

The z196 absolute address is used to access the z196 Configuration Array to get the z196 book’s memory physical address. This z196 book’s memory physical address indicates the book number, the MCU and the specific DIMM requested.
3.35 Hardware system area (HSA)

- 16 GB of HSA in any z10 EC model at power-on Reset
- Eliminates the installation pre-planning task for HSA configuration and eventual expansion size
- This size is able to contain:
  - 4 Channel Sub System (CSS)
  - 15 LPs in each CSS (total of 60 LPs)
  - Subchannel set-0 with 63.75k devices in each CSS
  - Subchannel set-1 with 64k devices in each CSS
- All these are designed to be activated and used with dynamic I/O changes

*Figure 3-36  Hardware system area*

**Hardware system area (HSA)**

HSA is like a central storage shared by all the activated logical partitions. HSA is built along a power-on Reset (POR) HMC operation. It contains the description of the channel subsystems, logical partitions, and I/O configuration. On top of that, it contains the LPAR LIC code and a copy of the CFCC operating system LIC code. It is always located in the central storage of the first book.

In z196 it always occupies 16 GB of central storage. As a result, in a z196, the installation planning for the HSA size is eliminated. Also, the preplanning for HSA expansion for reconfiguration is eliminated because HCD/IOCP will, via the POR process, always reserve the maximum size (16 GB) for the maximum I/O configuration possible. Note that there is room in the HSA for 7665 K (127.75 K x 60 LPs) sub channels (or UCWs).

*Note: HSA central storage is not included in the size of central storage that you purchase.*
3.36 Large pages

- Due to the "exploding" number of getmained pages:
  - Applications suffer a significant performance penalty because of TLB misses
  - Not allowed to increase the number of TLB entries due to performance constraints

- Solution: decrease the number of pages, making them bigger. Instead of 4 K, we have 1 M long-running memory access-intensive applications benefit
  - Optionally requested in IEASYSn member with the following parameter: LFAREA=nn% | nnM | nnG.
  - Large pages are treated as fixed pages and only available above 2 GB

Large page (1M) support

The reason that we are covering this subject is that the explosion of central storage size (lots of real addresses) and the explosion of virtual addresses (above the 2-G bar) is causing delays along the DAT translation process. Large pages minimize this effect. Traditionally, the z/Architecture virtual storage implementation defines:

- Pages of 4 K addresses
- Segments of 1 M addresses
- Regions of 2 G addresses

Virtual addresses are sliced into page, segment, and region numbers to be translated by the dynamic address translation (DAT) through the use of page tables, segment tables, and up to three region tables.

The translation look-aside buffer (TLB) is a fast memory located in each PU. In the z196, it has 512 entries. Each entry contains the real address associated with the most referenced 512 pages. The TLB consistently accelerates the DAT translation performance because with a TLB hit, DAT does not need to go to central storage in order to access the tables for translation.

However, the expected use of virtual addresses above the bar (2 G) is causing an explosion of pages. On the other hand, TLB sizes have remained relatively small due to low access time requirements and hardware space limitations.
TLB coverage today represents a much smaller fraction of an application's working set size pages, thus leading to a larger number of TLB misses. Applications can suffer a significant performance penalty resulting from an increased number of TLB misses, as well as the increased cost of each TLB miss.

So, if we are not allowed to increase the number of TLB entries, and virtual storage usage is exploding, the solution is to decrease the number of pages by making them bigger. That is, instead of 4 K, we have 1 M. Using large pages ensures that the TLB better represents the working set, and suffers fewer misses by allowing a single TLB entry to cover more address translations.

Exploiters of large pages are better represented in the TLB and are expected to perform better; long-running memory access-intensive applications particularly benefit. Short processes with small working sets see little or no improvement. The decision to use large pages must be based on the knowledge you gain from measuring memory usage and page translation overhead for a specific workload.

The large page size of 1 M addresses is optionally requested in the z/OS IEASYSnn member of SYS1.PARMLIB with the following parameter:

```
LFAREA=nn% | nnM | nnG.
```

**Note:** After the large page size is selected, it cannot be dynamically changed without an IPL. Large pages are treated as fixed pages and are never paged out. They are only available for 64-bit virtual private storage such as virtual memory located above 2 GB.
3.37 An I/O data flow tree analogy

I/O data flow for a read example

As shown in Figure 3-38, using a tree as an analogy for the z196, the “top leaves” (I/O channels) connect by a little branch, which connects to a larger branch and finally to the trunk. All I/O channels in one I/O card fork in the I/O card port. Four I/O card ports fork in an IFB-MP processor located in one I/O slot, as well. Two IFB-MPs fork in an HCA2-O (there are two IFB-MP links per HCA2-O fanout card). Eight HCA2-Os fork in one L4 cache. In all forks, traffic controllers allow, serially, all the confluent flows.

Let us follow the data flow experienced by a 4 KB DASD FICON read, through all the paths that the data might travel:

1. From a disk track to the DASD controller cache - and it was a cache miss.
2. From the DASD controller cache to the DASD controller host adapter buffer.
3. From the DASD controller host adapter buffer to the FICON switch director port buffer (moved within FICON data frames).
4. From the FICON director port buffer (moved within FICON data frames) to the channel I/O port buffer in one I/O card in one I/O cage.
5. From a channel I/O port buffer in one I/O card in an I/O cage, to an I/O port in the IFB-MP card.
6. From an I/O port in the IFB-MP card to an HCA2-O buffer.
7. From an HCA2-O buffer to the L4 cache.
3.38 Book to channel connectivity

In Figure 3-39, you may see all the copper cables and adapters that allow the connection between I/O channels and book elements as PUs and L4 cache. An I/O operation always moves data from the L4 cache (write) or to the L4 cache (read).

This picture does not show the optical coupling link to other books in other CPCs.

Refer to “Fanout cards in a book” on page 186 for more about Host Channel Adapter - HCA-C.

The other elements in this figure are described in the next foils.
3.39 Connecting books with I/O channels

A fanout card is a processor in charge of moving data in a z196 CPC between:
- The book and the I/O cage (I/O drawer) or
- An MVS LP and a Coupling Facility LP located in distinct CPCs.

Three types of fanout cards:
- HCA2-Copper are for links to I/O cages (or I/O drawers) in this CPC.
- HCA2-Optical are for links to external Coupling Facility in other CPCs (CIB type coupling links). Up to 150 meters.
- HCA2-Optical Long Reach (LR) are for links to Coupling Facility in other CPCs (CIB type coupling links). Up to 10 km unrepeated and 100 km repeated (with channel extension as DWDM).

Refer to “Fanout cards in a book” on page 186.

In Figure 3-40, you may see a simple view of the connections between the HCA and I/O cards.
3.40 FICON I/O card

I/O cards

Let us now cover I/O channels and I/O cards. I/O channels are specialized processors in charge of a dialog with another processor called Host Adapter located in an I/O controller. The purpose of such dialogue is to move data from or to the z196 memories (L4 caches) to or from an I/O device. This dialogue follows the rules of specific protocol such as FICON Express-8, through optical cables. A FICON Express-8 channel is able to move data at 200 MB/sec (roughly 2 Gb/sec), 400 MB/sec or 800 MB/sec depending on an auto negotiation done initially with the host adapter in the controller.

I/O channels are packaged in I/O cards, usually in groups of four. The I/O card is located in one I/O slot of an I/O cage or an I/O drawer.

Four I/O cards plus one IFB-MP half card make a domain. The IFB-MP is an InfiniBand processor in charge of multiplexing data coming from or to the domain's I/O cards.

As I/O cards continue to support higher data transfer rates to the devices, the connection between the I/O cards and the CEC cage needs to provide a higher data rate, as well. Refer to “InfiniBand interconnect technology” on page 153 for more information about this topic.

Figure 3-41 shows one FICON Express-8 I/O card with four FICON Express-8 channels.
3.41 FICON channel topics in z196

- In z196 only FICON Express-8 (800 MB/sec). FICON Express-4 is only available if carried over, on an upgrade.
- FICON Express-8 features:
  - Long wave (LX) 9 micron single mode fiber optic cables. Two options for unrepeated distances of up to 10 kilometers (6.2 miles).
  - Short wavelength (SX). For short distances. Uses 50 or 62.5 micron multimode fiber optic cables.
- FICON Extended Distance, new protocol for Information Unit (IU) pacing.

FICON Express-8 types

There are two types of FICON Express-8 channels:
- 10 KM LX – 9 micron single mode fiber long wavelength. Unrepeated distance - 10 kilometers (6.2 miles). Receiving controller must also be LX.
- SX – 50 or 62.5 micron multimode fiber short wavelength. Variable with link data rate and fiber type. Receiving controller must also be SX.

Each FICON Express-8 feature (I/O card) has four independent channels (ports) and can be configured to carry native FICON traffic or Fibre Channel (SCSI) traffic. LX and SX cannot be intermixed on a single I/O card feature, and the receiving control units must correspond to the appropriate LX or SX feature. The maximum number of FICON Express-8 I/O cards is 72 using three I/O cages (up to 288 channels).

FICON extended distance

An enhancement to the industry standard FICON architecture (FC-SB-3) helps avoid degradation of performance at extended distances by implementing a new protocol for “persistent” Information Unit - IU (also named frames) pacing.

Control units that exploit the enhancement to the architecture can increase the pacing count, the number of IUs (frames) allowed to be in flight from channel to control unit. Extended Distance FICON also allows the channel to “remember” the last pacing update for use on
subsequent operations to help avoid degradation of performance at the start of each new operation.

Improved IU pacing can help to optimize the utilization of the link (for example, to help keep a 800 MB/sec link fully utilized at 50 km) and provide increased distance between servers and control units. The requirements for channel extension equipment are simplified with the increased number of commands in flight. This may benefit z/OS Global Mirror (Extended Remote Copy, XRC) applications because the channel extension kit is no longer required to simulate (or spoof) specific channel commands. Simplifying the channel extension requirements may help reduce the total cost of ownership of end-to-end solutions. Extended distance FICON is transparent to operating systems and it applies to all the FICON traffic (CHPID type FC).

For exploitation, the control unit must support the new IU pacing protocol. The DS8000 series License Machine Code (LMC) level 5.3.1xx.xx supports such protocol. The channel defaults to current pacing values when operating with control units which cannot exploit extended distance FICON.

**z196 cascade directors**
As part of the high availability offered by z196, FICON Cascaded Director (connected switches) continues to be supported. Cascaded support is important for disaster recovery and business continuity solutions. It can provide high availability and extended distance connectivity, and has the potential for fiber infrastructure cost savings by reducing the number of channels for interconnecting the two sites.
### 3.42 I/O cage

We may have up to three I/O cages (nickname Cargo) in a z196 CPC, two in the A frame and one in the Z frame. Each has 36 slots:

- 28 are I/O slots filled with channel I/O cards.
- Four are slots filled with seven IFB-MP.
- Four are used for energy supply (DCA/CC).

Then, there is room for seven domains. Each domain occupies four I/O cards plus one IFB-MP (in a half I/O slot).

In a sense, we may say that the IFB-MP is a traffic cop managing the data traffic coming and going to each I/O card. It uses an InfiniBand algorithm. Refer to “Infiniband protocol” on page 238, for more information on such transmission algorithms.

The HCA-C also has a traffic cop role, managing the data traffic of two IFB-MPs into one L4 cache.
3.43 I/O drawer

Let us now explain the I/O drawer concept. It is a sort of I/O cage more flexible and with less channel capacity. I/O drawers provide increased I/O granularity and capacity flexibility and can be concurrently added and removed in the field—field replacement unit (FRU), an advantage over I/O cages, which also eases preplanning.

The z196 CPC can have up to six I/O drawers, two in the A frame and four in the Z frame. Then, z196 supports up to six I/O drawers, or two I/O cages or combinations of both. Installation of a third I/O cage requires a Request Price Quotation (RPQ), that is, a special request for manufacturing.

I/O drawers were first offered with the z10 BC. Each one supports two I/O domains (A and B) for a total of eight I/O card slots. Each I/O domain uses an IFB-MP card in the I/O drawer and a copper cable to connect to a host (HCA2-O).
3.44 Redundant I/O interconnect

If a book is removed or fails, the IFB-MPs has a redundant interconnect that allows each one to reach an HCA located in another book.

Redundant I/O Interconnect is a feature of the z196 CPC. It is implemented by the facilities of the IFB-MP card.

Each IFB-MP card is connected to an IFB jack located in the HCA2-C fanout card of a book. IFB-MP cards are half-high cards and are interconnected, thus allowing redundant I/O interconnect. If the IFB connection coming from a book ceases to function (such as when a book is removed), the interconnect path is used. Figure 3-45 shows a conceptual view of how Redundant I/O Interconnect is accomplished.

Normally, book 0 HCA2-C connects to the IFB-MP (A) card and services domain 0 I/O connections (slots 01, 03, 06, and 08). Book 1 HCA2-C/IFB connects to the IFB-MP (B) card and services domain 1 (slots 02, 04, 07, and 09).

If book 1 is removed, or if the connections from book 1 to the cage are removed, connectivity to domain 1 is maintained by guiding the I/O to domain 1 through the interconnect between IFB-MP (A) and IFB-MP (B).
### 3.45 Coupling Facility links

**External Coupling facility links**

External Coupling facility links connect MVS and Coupling Facilities located in different CPCs. There are two types of such links:

- HCA2-O fanout adapters in one book connected through optic fiber cables named Parallel Sysplex Infiniband (PSIFB) with HCA2-O located in another book in another z196 CPC. There are two ports per fanout. Each PSIFB may have several CHPIDs, four is recommended. There are two subtypes of HCA2-O, depending on the distance you need to connect. Refer to “Fanout cards in a book” on page 186, to get the types of HCA2-O.

- ISC-3 - Infiniband in the I/O cage connected through IFB-MP to HCA2-C (copper) in the book, and with fiber optics with another ISC-3 in another CPC, up to 100 km, 200 MB/sec.

**Internal Coupling facility links**

Internal Coupling Facility links connect MVS and Coupling Facilities located in the same CPC. They are called internal links (ICs). They are logical links in the CPC and data flows through the HSA central storage. There are up to 32 per CEC.
### 3.46 Infiniband protocol

#### InfiniBand

In 1999, two competing I/O standards called Future I/O (developed by Compaq, IBM, and Hewlett-Packard) and Next Generation I/O (developed by Intel, Microsoft, and Sun) merged into a unified I/O standard called InfiniBand. The InfiniBand Trade Association (IBTA) is the organization that maintains the InfiniBand specification and is led by a steering committee staffed by members of the before-mentioned corporations. The IBTA is responsible for compliance testing of commercial products, a list of which can be found at:

[http://www.infinibandta.org/itinfo/IL](http://www.infinibandta.org/itinfo/IL)

InfiniBand is an industry-standard specification that defines an input/output architecture used to interconnect servers, communications infrastructure equipment, storage, and embedded systems. InfiniBand is a true fabric architecture that leverages switched, point-to-point channels with data transfers up to 120 Gbps (12 GB/sec), both in chassis backplane applications, as well as through external copper and optical fiber connections.

InfiniBand is pervasive, low-latency, high-bandwidth interconnect, which requires low processing overhead and is ideal for carrying multiple traffic types (clustering, communications, storage, and management) over a single connection. As a mature and field-proven technology, InfiniBand is used in thousands of data centers, high-performance computing (HPC) clusters, and embedded applications that scale from two nodes up to a single cluster that interconnects thousands of nodes.
InfiniBand architecture

InfiniBand architecture defines a System Area Network (SAN) for connecting servers, I/O platforms, devices, serves as infra-structure in support of I/O and processor to processor communications. The architecture supports IP protocol to connect to Internet and remote servers. In addition, InfiniBand architecture defines a switched communication fabric for a high bandwidth, low latency environment. Much of processor load that is needed for communication I/O is off-loaded to InfiniBand hardware, thereby providing multiple concurrent operations without the protocol overhead.

InfiniBand is implemented in z196 to connect processor memory to I/O features and to provide coupling links to other servers in point-to-point configurations. InfiniBand links can be either copper or optical cables.

System z196 CPCs exploit InfiniBand for internal usage for connectivity from the Central Processor Complex (CPC) cage to the I/O cages. For external usage, Parallel Sysplex InfiniBand (PSIFB) links are used to interconnect System z196 and System z10 servers in a Parallel Sysplex environment.
3.47 z196 maximum number of channel per type

Table: Maximum number of channels per type

<table>
<thead>
<tr>
<th>Features</th>
<th>Avail</th>
<th>Maximum # of features</th>
<th>Maximum connections</th>
<th>Increments per feature</th>
<th>Purchase increments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESCON</td>
<td>Yes</td>
<td>16</td>
<td>240 channels</td>
<td>15 active</td>
<td>4 channels</td>
</tr>
<tr>
<td>FICON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FICON Express8</td>
<td>Yes</td>
<td>G</td>
<td></td>
<td>72*</td>
<td>268 channels</td>
</tr>
<tr>
<td>FICON Express4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISC-3</td>
<td>Yes</td>
<td>12</td>
<td>48 links</td>
<td>4 links</td>
<td>1 link</td>
</tr>
<tr>
<td>OSA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSA-Express3</td>
<td>Yes</td>
<td>G</td>
<td></td>
<td>24</td>
<td>96 ports</td>
</tr>
<tr>
<td>OSA-Express2**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GbE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSA-Express3</td>
<td>Yes</td>
<td>G</td>
<td></td>
<td>8</td>
<td>16 PCI adapters</td>
</tr>
<tr>
<td>OSA-Express2**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GbE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crypto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crypto Express3</td>
<td>Yes</td>
<td></td>
<td>8</td>
<td>16 PCI adapters</td>
<td>2 PCI adapters</td>
</tr>
</tbody>
</table>

Figure 3-48 Maximum number of channels per type

Maximum number of channels per channel type
In Figure 3-48, you can see the maximum number of channels per type in a z196 CPC.

Statement of Direction: The IBM zEnterprise 196 and the zEnterprise 114 are the last System z servers to support ESCON channels.

IBM plans not to offer ESCON channels as an orderable feature on System z servers that follow the z196 (machine type 2817) and z114 (machine type 2818). In addition, ESCON channels cannot be carried forward on an upgrade to such follow-on servers. This plan applies to channel path identifier (CHPID) types CNC, CTC, CVC, and CBY and to features #2323 and #2324.

Alternate solutions are available for connectivity to ESCON devices. For more information see:


ESCON channel
An ESCON channel executes commands presented by the standard z/Architecture I/O command set. It manages its associated link interface (link level/device level) to control bit transmission and reception over the optical medium (physical layer). On a write command,
the channel retrieves data from main storage, encodes it, and packages it into device-level frames before sending it on the link.

**Note:** The IBM zEnterprise 196 and IBM zEnterprise 114 are the last System z servers to support ESCON channels.

**Important:** The IBM ESCON Director 9032 (including all models and features) has been withdrawn from marketing. There is no IBM replacement for the IBM 9032.

Third-party vendors might be able to provide similar functionality to that of the IBM 9032 Model 5 ESCON Director.

**FICON modes and topologies**

System z supports the FICON channel to operate in one of three modes:

- FICON conversion mode (FCV)
- FICON native mode (FC)
- Fibre Channel Protocol (FCP)

**Note:** FICON Express2, FICON Express4, FICON Express8, and FICON Express8S features do not support FCV mode. FCV mode is available with FICON Express LX feature 2319 (carried forward only on System z10).

System z10 is the last server family to support FICON Express and FICON Express2.

zEnterprise 196 and zEnterprise 114 are the last systems to support FICON Express4 features. Review the usage of your installed FICON Express4 channels and, where possible, migrate to FICON Express8S channels.
3.48 z/OS discovery and auto-configuration (zDAC)

- It uses new capabilities in the z196 processor for Fabric discovery
- Reduces level of IT skill and time required to configure new I/O devices
- Ensures system (HCD host) and control unit definitions are compatible with each other
- Automatically discovers (via new options on HCD and HCM) and displays controllers and storage devices accessible to the system but not currently configured

Figure 3-49 zDAC functions

z/OS discovery and auto-reconfiguration (zDAC)

zDAC is a new function for z/OS running in a z196. It has the following properties:

- Provides automatic discovery for FICON disk and tape control units helping the HCD configuration task. Invoked and integrated into existing System z host configuration tools (HCD and HCM).
- Reduces level of IT skill and time required to configure new I/O devices.
- Ensures system (HCD host) and control unit definitions are compatible with each other.
- Automatically discovers (via new options on HCD and HCM) and displays controllers and storage devices (for inclusion or exclusion) accessible to the system but not currently configured, and proposes host definition values:
  - For discovered logical control units, explore for defined logical control units and devices. Also discover new devices for existing logical control units.
  - Compare discovered logical control units and devices against those configured previously, indicating whether the element is new, that is, no existing devices/LCU defined in the target IIODF.
  - Add missing logical control units and devices to the configuration, proposing control unit and device numbers, and proposing new paths to reach them.
  - Channel paths chosen using an algorithm to minimize single points of failure.
  - Addition of a subchannel set of 64K devices to the existing two subchannel sets.
- With zDAC it is suggested to implement FICON DCM (z/OS dynamic channel path management) to help improve I/O performance.
3.49 WWPN and fabrics discovery

The WWPN and fabrics discovery provides the following support and has the following characteristics:

- Adds support to allow assignment of WWPN to physical FCP ports. Currently LIC assigns worldwide port names (WWPN) to virtual FCP ports.
- Allows WWPN Prediction Tool to calculate WWPN for virtual and physical ports ahead of system installation time.
- Customer value - Can keep their SAN configuration after replacement of an FCP Channel card. Improved system installation. Simplifies FCP channel card replacement procedures.
- Transparent to operating systems.
- WWPN tool available for download from Resource Link®.
3.50 **zDAC software and hardware requirements**

- z196 CPC with FICON Express8 or FICON Express4
- Switch/director attached fabric (no direct attachment).
  - Brocade Firmware Version: V6.2.0e
  - McData Firmware Version: 09.09.00
- z/OS V1R12 (and at least one logical partition for Dynamic I/O capability)
- First exploiter is IBM System Storage DS8700.
  DS8000® licensed machine code level 6.5.15.xx (bundle version 75.15.xx.xx), or later
  - See the IBM PSP buckets

Figure 3-51  zDAC software and hardware requirements

*zDAC software and hardware requirements*

In Figure 3-51, we see all the z/OS, DS8700 and channel requirements to implement zDAC.
### 3.51 zDAC policy in HCD

**Figure 3-52   HCD auto configuration policies panel**

<table>
<thead>
<tr>
<th>z/OS V1.12 HCD</th>
<th>Discovery and Autoconfiguration Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify autoconfiguration options. Then, press Enter to start the discovery process.</td>
<td></td>
</tr>
<tr>
<td>Autoconfiguration is based on 1 1. Active IODF 2. Currently accessed IODF</td>
<td></td>
</tr>
<tr>
<td>Show proposed definitions .. 1 1. Yes 2. No</td>
<td></td>
</tr>
<tr>
<td>Scope of discovery .. . . . 1 1. New controllers only 2. All controllers 3. Controller containing CU _____ +</td>
<td></td>
</tr>
<tr>
<td>Force full mode discovery .. 2 1. Yes 2. No</td>
<td></td>
</tr>
<tr>
<td>Target IODF name .. 'SYS6.IODF04.WORK' +</td>
<td></td>
</tr>
<tr>
<td>F1=Help F2=Split F3=Exit F4=Prompt F5=Reset F9=Swap</td>
<td></td>
</tr>
<tr>
<td>F12=Cancel</td>
<td></td>
</tr>
</tbody>
</table>

**zDAC policy in HCD/HMC**

Figure 3-52 shows the HCD zDAC primary panel where the system programmer defines the zDAC policy.

To reach this panel, select option 6 in the HCD primary menu.

Here you may select a specific IODF and also select the scope of the discovery, as such:

- New controllers only
- All controllers
- Specific set of controllers
3.52 zDAC policy

Through HCD or HCM the customer establishes a policy for the discovery and auto configuration operation which can:

- Limit the scope of the discovery
- Limit the proposal information
- Indicate the desired number of paths to discovered logical control units
- Indicate the method used for device and logical control unit numbering

Figure 3-53  zDAC policy
3.53 zDAC discovered HCD panel

Figure 3-54  HCD Discovered New or Changed Controller List panel

HCD discovered new panel
Figure 3-54 shows the HCD panel where the system programmer sees the defined logical control units not defined in the HCD.

HCD displays the result in the Discovered New or Changed Controller List. Only discovered controllers are returned, which are reachable from all target systems that have partitions defined in the LP group referenced by the AUTO_SUG_LPGROUP policy keyword.
3.54 zDAC proposed HCD panel

HCD proposed control unit list panel

With Figure 3-55, the system programmer may agree to add the missing logical control units and devices to the configuration, by proposing control unit and device numbers, and by proposing new paths to reach them.

In the View Control Unit Definition panel (Figure 3-55), note the following:

- The Serial # now indicates the Serial Number of the controller.
- Processor SCZP301 is now connected with static CHPIDs and two dynamic CHPIDs.
3.55 Logical channel subsystem

Logical channel subsystem (LCSS)
The channel subsystem is made of 256 I/O channels. Sometimes it is also called logical channel subsystem. It controls communication of internal (as IC and IQD) channels and external (as FICON) channels to control units and devices. The configuration definitions of the LCSS (at HCD) define the operating environment for the correct execution of all system input/output (I/O) operations. The channels permit transfer of data between main storage (L4 cache) and I/O devices (through controllers) or other servers under the control of a channel program. The LCSS components allow channel I/O operations to continue independently of other operations within the CPs.

An important piece of the I/O z/Architecture is the SAP, which is a PU in charge of guaranteeing the start and the end of an I/O operation. It is a mainframe-unique feature that frees many processor cycles during I/O operations.

Multiple logical channel subsystems
An LCSS, by definition, is made of up to 256 channels and does not include the SAPs. A z196 EC allows up to four LCSS, and makes it possible to have up to 1024 channels, allowing performance benefits for larger installations.

CHPID
Within the LP, the channel is accessed by the operating system through the one-byte CHPID, as defined in z/Architecture. This one-byte CHPID limits the maximum number of channels.
that can be accessed by a copy of the operating system in the LP. Due to compatibility reasons this number cannot be extended. In order to allow more than 256 channels per CPC, the concept of LCSS was introduced. In other words, CHPIDs are unique within an LCSS ranging from 00 to FF. However, the same CHPID number range is used again in the other three LCSSs.

Then, each logical partition (LP) and its operating system can only access one LCSS and consequently up to a maximum of 256 channels. Different LPs (up to 15) may share the same LCSS or have a different LCSS, but there can be just one LCSS per LP.

Outside the LP scope, as in an HCD definition, the channel is identified by the CHPID qualified by the LCSS ID, as shown in this IOCP statement:

```plaintext
CHPID PATH=(CSS(0),80),SHARED,PARTITION=((LP1,LP2,LP3), (=)),
```

The CHPID 80 from LCSS 0 is shared (MIF-shared) between logical partitions LP1, LP2, and LP3.

As shown in Figure 3-56 on page 249, in the same server (CPC) you can have several channels (from different LCSSs) with the same CHPID. The CHPIDs 80, 81, 82 and 83 are repeated, but in a distinct LCSS. Thus, a single operating system instance still has a maximum of 256 CHPIDs, but the server as a whole can have more than 256 CHPIDs.

Logical partition (LP) summary

A given logical partition (LP) is associated with a single LCSS, and a single LCSS has a maximum of 256 CHPIDs. Multiple LPs may be associated with a given LCSS, as follows:

- Each LCSS may have from one to 256 channels.
- Each CHPID is qualified by its LCSS ID.
- Each LCSS can be configured with 1 to 15 logical partitions.
- The four LCSSs can be configured to 60 logical partitions per CPC.

Figure 3-56 on page 249 illustrates the relationship between LPs, MIF IDs, LCSSs, CHPID, and PCHID. The concept of PCHID is illustrated in Figure 3-58 on page 253. The concept of MIF IDs is explained in “LP ID and MIF ID concepts” on page 251.

The I/O subsystem continues to be viewed as a single input/output configuration data set (IOCDS) across the entire system with multiple LCSSs. Refer to “CSS configuration management” on page 384 for more information about IOCDS. Only one hardware system area (HSA) is used to describe, through control blocks, the multiple LCSSs.

The channel definitions of an LCSS are not bound to a single book. An LCSS may define channels that are physically connected to all HCA2-C of all books in any multibook z10 EC model.
3.56 LP ID, MIF ID, and spanning concepts

LP ID and MIF ID concepts
In this section we explain the concepts underlying Logical Partition ID (LP ID) and Multiple Image Facility ID (MIF ID) and how they relate.

Logical partition ID (LP ID)
An LP ID is an identification of the logical partition, together with the logical partition name (LPname). LP ID is defined in the LPAR profile in the HMC and LPname in HCD. When LPAR was announced, the LP ID was defined with a one-half byte length from X’1’ to X’F’. Originally, this limited the maximum number of LPs to 15.

The LP ID is used by several hardware and software functions, as explained here:
- A FICON channel identifies itself to the I/O control unit in a multi-image facility (MIF) setup, when the same channel may serve several LPs, by using the four-bit LP ID.
- For the CTC control unit logical address of the remote CTC, when an MIF channel path is used.
- As data moved into storage by the STORE CP ID (STIDP) instruction.
- To identify the z/OS system that sends the path group ID CCW to an I/O controller.

z196 LP support
With the z196 (and z10), the maximum allowed number of LPs is now 60. To accommodate this, a new LP ID field is introduced with two hex characters (one byte), from X’00’ to X’3F’.
MIF Image ID

However, all functions depending on an LP ID still expect a four-bit field. For example, the FICON protocol for implementing EMIF still works with an LP ID from X’1’ to X’F’. The solution is the creation of the MIF ID as a replacement. The MIF ID (with the contents of the old LP ID) is used instead of the LP ID. To maintain the uniqueness of a MIF ID, it is qualified by the LCSS.

Then, the MIF Image ID is a number that is defined through HCD, or directly via the IOCP through the RESOURCE statement. It is in the range ‘1’ to ‘F’ and is unique within an LCSS, but it is not unique within z196. Multiple LCSSs may specify LPs with the same MIF Image ID.

Note the following summary points:

- The logical partition names are specified in the I/O definition process (HCD or IOCP), and must be unique for the logical partition across all LCSSs in the z196.
- The logical partition MIF ID is specified in the I/O definition process (HCD or IOCP), and must be unique x’0’ to ‘F’ for all logical partitions across each CSS in the z196.
- The logical partition ID is specified (by the user) in the z196 image profile (for the LP in the HMC), and must be unique x’00-3F’ for the logical partition across all CSS in the z196.

In Figure 3-57 on page 251, LP 1 has a MIF ID of 1. In another LCSS, LP 14 has a MIF ID of 1, as well.

Spanning

A spanned channel is a channel belonging to more than one LCSS. Using spanned channels decreases the number of channels needed in your installation. With the z196, the majority of channel types can be defined as spanned. The exceptions are:

- ESCON channels defined with CHPID type CVC or CBY
- The following channels can be shared but not spanned:
  - ESCON channels defined with CHPID type CNC or CTC
  - FICON channels defined with CHPID type FCV

Comparing spanned and shared channels

The Multiple Image Facility (MIF) allows channels to be shared among multiple logical partitions in a server.

- Shared channels can be shared by logical partitions within a same LCSS.
- Spanned channels can be shared by logical partitions within and across LCSSs.

All other channels can be shared and spanned:

- FICON Express when defined as CHPID type FC or FCP
- FICON Express4 and FICON Express8
- OSA-Express and OSA-Express2
- Coupling links channels in peer mode: IISC-3
- Internal channels: IC (ICF link) and HiperSockets

HiperSockets

A spanned channel occupies the same CHPID number in all LCSSs in which it is used. For example, if a HiperSockets channel is CHPID 2C in LCSS 0, it must be the same CHPID number in LCSS 1 (if LCSS 1 is used, of course, and if that HiperSockets is also defined in LCSS1). A HiperSockets that connects LPs in different LCSSs must be spanned.
3.57 Physical channel ID (PCHID)

Physical channel IDs (PCHID)

Before the introduction of the PCHID in zSeries, each CHPID was associated one-to-one, with each channel also being a physical identification of such a channel. With PCHID, the CHPID is just a logical name or a nickname for the channel, and it no longer directly corresponds to a hardware channel. On the other hand, the 12-bit PCHID number identifies even the physical slot location within an I/O card in the cage (or I/O drawer).

This implementation increases the flexibility of defining I/O channel configurations. You may keep the CHPID numbers (for example, 01, 21, 31, 41 for DASD) even when migrating to another server with other channels associated with other PCHIDs.

CHPIDs are not pre-assigned to channels. It is the customer's responsibility to assign only the CHPID numbers and through the use of the CHPID Mapping Tool (CMT) utility program the PCHIDs are assigned. The PCHID assignment can also be done directly with the HCD/IOCP program (which is not recommended). Assigning CHPID means that the CHPID number is associated with a physical channel port location (PCHID) and an LCSS containing that channel. The CHPID number range is still from ‘00’ to ‘FF’ and must be unique within an LCSS. Any CHPID not connected to a PCHID fails validation when an attempt is made to build a production IODF or an IOCDS.

Figure 3-58 shows the front view of the first I/O cage (at the bottom of the A frame), including some I/O cards in slots 01 to 08, and the PCHID numbers of each port.
Channels
A channel is identified by a three-digit PCHID, or physical channel identifier. A PCHID number is defined for each potential channel interface. As an example, in the following IOCP statement, the CHPID 80 of LCSS 0 corresponds to the PCHID 0140:

```
CHPID PATH=(CSS(0),80),SHARED,PARTITION=((LP1,LP2,LP3)=(X),SWITCH=61,TYPE=FC,PCHID=0140
```

For ESCON channels, each ESCON I/O card has up to 16 adapters or channel interfaces. 16 PCHID numbers are reserved for each I/O adapter slot in each I/O cage. Not all I/O cards provide 16 channels (FICON I/O card has four channels), but 16 PCHID numbers are reserved for each possible I/O slot.

**Note:** The PCHID numbers allocated to each I/O adapter and port on that adapter are fixed and cannot be changed by a user.

Each enabled I/O channel has its own PCHID number, which is based on the following:
- Its I/O port or channel interface (also called the **connect number**) in the I/O card
- Its I/O card location
- Its I/O cage (or I/O drawer)

Table 3-1 shows the PCHID numbering scheme.

<table>
<thead>
<tr>
<th>Cage</th>
<th>Front PCHID ##</th>
<th>Rear PCHID ##</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Cage 1</td>
<td>100 - 11F</td>
<td>200 - 21F</td>
</tr>
<tr>
<td>I/O Cage 2</td>
<td>300 - 31F</td>
<td>400 - 41F</td>
</tr>
<tr>
<td>I/O Cage 3</td>
<td>500 - 51F</td>
<td>600 - 61F</td>
</tr>
<tr>
<td>Server Cage</td>
<td>000 - 03F</td>
<td>reserved for ICB-4s</td>
</tr>
</tbody>
</table>

As mentioned, it is the customer’s responsibility to perform these assignments of CHPID to PCHID by using HCD and IOCP definitions and the assistance of the CHPID Mapping Tool (CMT). Using CMT provides an IOCP source that maps the defined CHPID to the corresponding PCHID of the server.

**Note:** There is no absolute requirement to use CMT; you can assign CHPID to PCHID directly in an IOCP source or through HCD. However, this is a very cumbersome process for large configurations.

If you choose to perform a manual assignment of CHPID to PCHID (using HCD or IOCP), it is your responsibility to distribute CHPID among the physical channel card ports (PCHID) for availability and performance. The objective of CMT is to assist in performing these tasks.
3.58 Association between CHPID and PCHID

A z196 supports up to 1024 physical channels (PCHID). In order to be used for I/O operation by a z/OS, the PCHID must be mapped to a CHPID. Each CHPID is uniquely defined in an LCSS and mapped to an installed PCHID by the customer through HCD, IOCP, or the CHPID Mapping Tool. A PCHID is eligible for mapping to any CHPID in any LCSS.

For internal channels, such as IC links and HiperSockets, CHPIDs are not assigned to a PCHID.

I/O channels
I/O channels can be shared between logical partitions by including the partition name in the partition list of a Channel Path ID (CHPID). I/O configurations are defined by the I/O Configuration Program (IOCP) or the Hardware Configuration Dialog (HCD) in conjunction with the CHPID Mapping Tool (CMT). The CMT is an optional, but strongly recommended, tool used to map CHPID onto Physical Channel IDs (PCHID) that represent the physical location of a port on a card in an I/O cage.

Operating systems
IOCP is available on the z/OS, z/VM and z/VSE operating systems, and as a standalone program on the z196 HMC hardware console. HCD is available on z/OS and z/VM operating systems.
3.59 Comparison between System z servers

Comparison between system z servers

Figure 3-60 shows a comparison of the current implementation on LP ID and LP names with the previous ones (z800 and z900). We divide the System z family into two groups: previous (z800 and z900) and modern (z990, z890, z9 EC, z9 BC, z196). Following is the comparison of the definition terms used with System z9 and zSeries servers:

**LP name**

In previous servers, this name is defined through HCD or in the IOCP logical partition name in the RESOURCES statement in an IOCP. The names must be unique across the server.

In modern servers, this is the same.

**LP ID**

In previous servers, the logical partition identifier is used as the four-bit digit of the operand stored by the Store CP ID instruction. This value must be in the range of 0 to F. The LP identifier must be unique for each active LP. The value is assigned on the General Page of the Image Profile for the LP in the Hardware Management Console (HMC), in the LPAR number parameter in HCD, and in the RESOURCES statement in IOCP.

In modern servers, the LP ID is a value in the range of 00 to 3F. It is assigned in the image profile through the support element (SE) or the HMC. This identifier is returned by the Store CP ID (STIDP) instruction. It is unique across the z9 EC server, and is also referred to as the user logical partition ID (UPID).

**MIF ID**

In previous servers, this is the same entity as the LP ID.
In modern servers, the MIF ID is introduced because these servers allow more than 15 LPs. Therefore, the LP ID cannot be used to implement MIF channels, CTCA protocol, or the I/O control unit path group ID. Then, the MIF ID ranges from X’1’ to X’F’ as demanded by such protocols, and it is unique within an LCSS. It is specified in HCD/IOCP in the LP number option.

**I/O component comparison**

Table 3-2 lists the number of I/O components supported on System z9 and zSeries servers.

<table>
<thead>
<tr>
<th></th>
<th>z800 and z900</th>
<th>z890</th>
<th>z990</th>
<th>z9 BC</th>
<th>z9 EC, z10 and z196</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LCSSs</strong></td>
<td>1 per server</td>
<td>2 per server</td>
<td>4 per server</td>
<td>2 per server</td>
<td>4 per server</td>
</tr>
<tr>
<td><strong>Partitions</strong></td>
<td>15 per server</td>
<td>15 per LCSS</td>
<td>15 per LCSS</td>
<td>15 per LCSS, up to 30(^a) per server</td>
<td>15 per LCSS, up to 60 per server</td>
</tr>
<tr>
<td><strong>CHPID</strong></td>
<td>256 per LCSS</td>
<td>256 per LCSS</td>
<td>256 per LCSS</td>
<td>256 per LCSS, up to 512 per server</td>
<td>256 per LCSS, up to 1024 per server</td>
</tr>
<tr>
<td><strong>Devices</strong></td>
<td>63 K per server</td>
<td>63 K per LCSS</td>
<td>63 K per LCSS</td>
<td>63.75 K per LCSS(^b), up to 127.5 K per server</td>
<td>63.75 K per LCSS(^b), up to 255 K per server</td>
</tr>
</tbody>
</table>

---

\(^a\) z9 BC model R07 (capacity setting A01 only) supports a maximum of 15 logical partitions per server.

\(^b\) Multiple subchannel sets are supported.
3.60 IOCP statements example

In the example shown in Figure 3-61, the first appearance of the new parameters for the modern servers is shown in bold font from the HCD-produced IOCP. As when the LP is defined in the RESOURCE statement, it is associated with the ID of the LCSS, such as LCSS(0).

The former LP ID is now the MIF ID, such as (LP1,1). The current LP IDs are defined through HMC panels.

MAXDEV keyword
MAXDEV reserves space in the HSA, based on the maximum number of UCWs (subchannels), to allow Dynamic I/O Configuration. In a z196 server there is no longer any need to reserve space in the HSA because a full 16 GB is reserved, so the use of MAXDEV is unnecessary.

Defining CHPID
When defining a CHPID, it must be qualified by the LCSS ID, such as CSS(0), 80, 81, 90, 91. The CHPID must be mapped to a PCHID, as shown in the CHPID statement. This mapping can be done by using the CHPID mapping tool.

Recommendation: Use HCD instead of coding IOCP statements.
3.61 Configuration definition process

Assigning PCHIDs

<table>
<thead>
<tr>
<th>HCD</th>
<th>IODF no PCHIDs</th>
<th>CF Report or H/W Config Report for your order (CCN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Create IODF for your server without PCHIDs</td>
<td>IODF source no PCHIDs</td>
<td>CHPID Mapping Tool</td>
</tr>
<tr>
<td>2. Create IODF (validated work IODF) - HCD option 2.12 = Build validated work I/O definition file</td>
<td>IOCP source with PCHIDs</td>
<td>4. Run CHPID mapping tool. It will produce an IOCP source with PCHIDs assigned</td>
</tr>
<tr>
<td>3. Create IOCP source without PCHIDs - HCD option 2.3 = Build IOCP input data set</td>
<td>IOCP source with PCHIDs</td>
<td></td>
</tr>
<tr>
<td>5. Import IOCP source with PCHIDs into IODF - HCD option 5.1 = Migrate IOCP/OS data, Migrate option 3 (PCHIDs)</td>
<td>IODF with PCHIDs</td>
<td></td>
</tr>
<tr>
<td>6. Create a production IODF - HCD option 2.1 = Build production I/O definition file</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-62 Configuration definition process

CHPID mapping tool (CMT)

Figure 3-62 shows a diagram containing the suggested steps needed to define a new build for a z196, z10 EC, z9 EC, z9 BC, z990, or z890 I/O configuration.

The old servers had fixed CHPID assignments. The modern servers (z990, z9 EC, z10 and z196) use a concept called Channel CHPID Assignment where the CHPIDs are not permanently assigned to a particular card slot, but instead are assigned as needed.

It is strongly recommended that you use the CHPID Mapping Tool to configure the CHPID-to-PCHID assignments. The IODF that you created first should not have PCHID assignments. CMT reads an IOCP source file sent by HCD with no PCHID. CMT also reads the CF report and HW Config report (documents produced by IBM) and produces an IOCP source file with the PCHID assigned. This data set is sent to HCD to finalize the configuration in the IODF.

The CHPID Mapping Tool provides a way to customize the CHPID assignments for a z196 system to avoid attaching critical channel paths to single points of failure, such as two channels from the same I/O card reaching the same I/O control unit. This tool should be used after the server order is placed and before the system is delivered for installation.

This mapping tool can also be used to remap CHPID after hardware upgrades that increase the number of channels. The tool maps the CHPID from an IOCP file (usually generated by HCD) to Physical Channel Identifiers (PCHID) which are assigned to the I/O ports. As
mentioned, the PCHID assignments are fixed and cannot be changed. The CHPID Mapping Tool is available from Resource Link as a standalone PC-based program.

**Channel path identifier (CHPID)**

Existing software code works with single-byte CHPID numbers, producing the limitation of 256 CHPIDs. The difficulty is to extend this number while maintaining compatibility with existing software. The 256 maximum CHPID number is reflected in various control blocks in operating systems, software performance measurement tools, and even some application code.

The new architecture provides multiple sets of channel definitions, each with a maximum of 256 channels. Existing operating systems would be associated with one logical channel subsystem (LCSS), and work with a maximum of 256 CHPIDs. Different logical partitions can be associated with different channel subsystem definitions. Thus, a single operating system instance (using existing code) still has a maximum of 256 CHPIDs, but the server as a whole can have more than 256 CHPIDs. To exploit more than 256 channels, it is necessary to have multiple operating images (in multiple LPs).

**Physical channel identifier (PCHID)**

A PCHID reflects the physical identifier of a channel-type interface. A PCHID number is based on the I/O cage location, the channel feature slot number, and the port number of the channel feature. A CHPID does not directly correspond to a hardware channel port on a z9 EC server, and may be arbitrarily assigned. A hardware channel is now identified by a PCHID.

You can address 256 CHPIDs within a single channel subsystem. That gives a maximum of 1024 CHPIDs when four LCSSs are defined. Each CHPID is associated with a single channel. The physical channel, which uniquely identifies a connector jack on a channel feature, is known by its PCHID number.

**Using the CMT**

The result of using the CMT tool is an IOCP deck that maps the defined CHPID to the corresponding PCHID of the server. Although there is no requirement to use the mapping tool—you can assign CHPID to PCHIDS directly in IOCP decks or through HCD, but this is a very cumbersome process for larger configurations—it is easier and more efficient to use the tool to do channel mapping.

If customers choose to perform manual assignment of CHPID to PCHID (using HCD or IOCP), it is their responsibility to distribute CHPIDs among the physical channel cards (PCHID) for availability (to avoid single point of failures) and performance. The objective of the tool is to help in performing these tasks.

Figure 3-62 on page 259 shows a diagram containing the suggested steps that an installation can take in defining a new z9 EC I/O configuration.
3.62 Channel availability features

There are two channel features in z196 that improve full system availability:

- **System-initiated CHPID reconfiguration function**
  - Designed to reduce the duration of a repair action and minimize operator interaction when an ESCON or FICON channel, an OSA port, or an ISC-3 link is shared across logical partitions.

- **Multipath IPL**
  - Designed to help increase availability, to help eliminate manual problem determination when executing an IPL.

---

Figure 3-63  Channel availability features

Channel availability features

There are two channel features in z196 that improve full system availability, as follows:

*System-initiated CHPID reconfiguration function does the following:*

- This function is designed to reduce the duration of a repair action and to minimize operator interaction when an ESCON or FICON channel, an OSA port, or an ISC-3 link is shared across LPs. When an I/O card is replaced for a repair, it usually has some failed and some still functioning channels. To remove the card, all channels need to be configured offline from all LPs sharing them.

- Without system-initiated CHPID reconfiguration, the IBM customer engineer (CE) must contact each LP z/OS operator and have the channels set offline, and then after the repair, contact them again to configure channels back online.

- With system-initiated CHPID reconfiguration, the Support Element sends a signal to the SAP that a channel needs to be configured offline. IOP determines all the LPs sharing that channel and sends an alert to all z/OS systems in those LPs. z/OS then configures the channel offline without any operator intervention. This is repeated for each channel on the card.

- When the card is replaced back, the Support Element sends another signal to SAP for each channel. This time the IOP alerts z/OS that the channel should be configured back online. This is designed to minimize operator interaction to configure channels offline and online.
Multipath IPL function

During an IPL, if an I/O operation to the IPL device fails due to an I/O error, multipath IPL retries the I/O using an alternate path. This function is applicable to ESCON and FICON channels. z/OS supports multipath IPLs.
3.6.3 Introduction to MIDAW

Modified indirect data address word (MIDAW) facility

z/Architecture supports a new facility for indirect addressing, the Modified Indirect Data Address Word (MIDAW) facility, for both ESCON and FICON channels and for z9, z10 EC and z196 CPCs. The use of the MIDAW facility, by applications that currently use data chaining, may result in improved channel throughput in FICON environments. This facility may be heavily used by VSAM media manager.

MIDAW facility is exploited by z/OS V1R7 plus APARs and I/O controller awareness for the DS8000.

Results of internal DB2 table scan tests with extended format data sets on the z9 EC with the MIDAW facility and the IBM TotalStorage DS8000 yielded the following results when using FICON Express4 operating at 400 MB/sec on a z9 EC compared to FICON Express2 operating at 200 MB/sec in VSAM cluster:

- A 46% improvement in throughput for all reads (270 MB/sec versus 185 MB/sec)
- A 35% reduction in I/O response times

For more details about MIDAW performance, refer to “MIDAW performance results” on page 273.
I/O operations summary

An I/O operation includes a dialog between a FICON (or ESCON) channel microprocessor located on the FICON feature card, as shown at point (2) in Figure 3-64 on page 263.

The objective is the transfer of data between the server’s real memory and a device media, managed by that I/O controller. The channel is in command of the I/O operation, requiring certain actions from the controller. To know what to order, the channel accesses, in real storage, an already-written channel program, as shown at point (0) in Figure 3-64 on page 263. As an analogy, the processor executes programs comprised of instructions, and the I/O channel executes channel programs composed of channel command words (CCWs). That is, a channel program describes to the channel the actions it needs to order from the I/O controller for an I/O operation. A CCW is the basic element of a channel program.

The CCW contents are explained here (refer to point (0) in Figure 3-64 on page 263):

- The first 8 bits (CMD) contain the command code; for example, X'02' is a read and X'01' is a write.
- Flags (FLGS) indicate options.
- Byte count (COUNT) indicates the size of the physical block to be transferred by this CCW. For a write, it is the number of bytes to be stored in the device. For a read, it is the number of bytes to be transferred to storage.
- Data address (@DATA) specifies the address in storage for the physical block read or write. This storage area is also called the I/O buffer.

For more information about CCWs, refer to “Channel command word (CCW) concept” on page 265.
3.64 Channel command word (CCW) concept

- **CCW Define Extent**
  - Used for security, integrity, DASD cache management

- **CCW Locate Record**
  - Tells the location of the physical record (cyl, track and record) in the DASD device

- **CCW Read 4 KB,flags,10000**
  - Reads 4096 bytes into contents of storage address 10000
  - Flag bits are used for indicating some options

---

### Channel command word (CCW) concepts

Following are definitions regarding the use of CCWs.

#### DEFINE EXTENT CCW

The DEFINE EXTENT CCW is always the first one in the channel program, and it is added by the input output supervisor (IOS). This CCW deals with security, integrity, and DASD cache control of the channel program.

#### LOCATE RECORD CCW

The LOCATE RECORD CCW indicates to the channel the physical address (cylinder, track, and physical block number) in the DASD device of the physical block to be read or written. This information is passed by the channel to the controller.

#### READ CCW

The READ CCW informs you that the physical block described by the LOCATE RECORD CCW is to be moved from the DASD device to storage. Two informational items are key: the address in storage, and the size of the physical block (for checking purposes).

Instead of the READ CCW, you can have a WRITE CCW that informs you that the physical block already prepared in storage is to be moved to a DASD device location, as described by the LOCATE RECORD CCW. Two informational items are also key: the address in storage, and the size of the physical block to be written in the device track.
3.65 CCWs and virtual storage - IDAW Concept

As shown in Figure 3-66, an 8 KB output I/O buffer is aligned in a page boundary starting at virtual address 12 M. The access method builds the following CCW:

```
CCW write, 8-KB, flags, 12-M
```

The access method is software and only understands virtual addresses. However, the channel cannot reach the DAT that is located in the processor storage page within each PU. The channel does not understand virtual addresses, only real addresses. Therefore, the z/OS component I/O driver performs the translation.

In this example, the page starting at the 12 M virtual address is located in the frame starting at real storage 200 KB. The next page, the one at 12 M plus 4 K, is located in the frame starting at real storage 100 KB. As you can see, there is no I/O buffer continuity in real storage.

If the I/O driver only replaces the 12 M by 200 KB, the channel moves the 8 KB from storage starting at frame 200 KB and continuing into frame 204 KB, which is clearly a mistake.

To avoid such a problem, the concept of *indirect addressing* was introduced. If the CCW flag bit 13 is on, in the CCW, then the channel understands that the CCW-address field, instead of containing a real address of the I/O buffers, contains the real address of a list of double words, called indirect data address words (IDAWs).

---

**Figure 3-66  IDAW concept**

**CCWs and virtual storage - IDAW concept**

Virtual Storage Pages

Real Storage Frames

| IDAW 0 | 200 K |
| IDAW 1 | 100 K |
Each IDAW contains a real address designating a 4 KB frame. When the first 4 KB frame is exhausted, the channel keeps the I/O data transfer to the real address in the next IDAW in the list. The list is called an *indirect data address list* (IDAL). Therefore, in our numeric example, the first IDAW points to real address 200 K and the second points to real address 100 K.

Although the IDAW design allows the first IDAW in a list to point to any real address within a page frame, subsequent IDAWs in the same list must point to the first byte in a page frame. Also, all but the first and last IDAW in a list must deal with complete 4 KB units of data.

Observe that the CCW-address field only has 31 bits (up to 2 GB addresses). In order to have an I/O buffer above the bar (higher than 2 GB real address), the IDAWS must be always used for such I/O, because each IDAW has 64 bits in order to point beyond 2 GB.

**CCW command chaining**

An I/O physical block contains logical records. Its size is determined by the access method parameter installation-defined BLKSIZE. On DASD or on tape, each physical block is separated from others by gaps.

Generally speaking, each read or write CCW is able to transfer only one physical block. However, it is possible in the same channel program to transfer more than one physical block. To implement this, a flag bit called the *command chain bit 9* is set.

The CPU default design is “when you finish one instruction, execute the next”. The channel default design is “when you finish one CCW, quit (send an I/O interrupt informing the end of the channel program execution)”. The command chaining flag is needed in order to make the channel execute the next CCW. Therefore, all CCWs in a channel program (except the last one) should have the command chaining bit on. With sequential I/O processing it is very common, for performance reasons, to transfer several physical blocks in just one channel program by using the command chaining facility.

**Data chaining**

Data chaining is a special case of command chaining. It is activated by flag bit 8 in the CCW. *Data chaining* means that several CCWs, in sequence, with this flag on, operate on the same physical record. Basically there are two reasons for this implementation:

- To read and write a physical block greater than 64 KB.

  Keep in mind that the byte count in the CCW only allows 64 KB (it has 16 bits). Look at the following channel program with virtual addresses:

  
  
<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCW Write 64 KB, flag 8 On, 10 M</td>
<td></td>
</tr>
<tr>
<td>CCW Write 64 KB, flag 8 On, 10 M + 64 K</td>
<td></td>
</tr>
<tr>
<td>CCW Write 64 KB, flag 8 On, 10 M + 128 K</td>
<td></td>
</tr>
<tr>
<td>CCW Write 64 KB, flag 8 Off, 10 M + 192 K</td>
<td></td>
</tr>
</tbody>
</table>

  As you can see, this channel program is writing just one physical block of 256 KB located at the 10 M virtual address.

- To read and write a “spread physical record (also called scatter-read or scatter-write)”.

  As an example, with 3390 DASD, the physical record or block is usually comprised of two items: a count with 8 bytes containing control information (transparent to the application), and data which contains the data itself. If an access method wants to read the specified count to an internal specific storage area and the data to the I/O buffer, it can produce the following virtual channel program:

  
<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCW Read 8, flag 8 On, 20 M</td>
<td></td>
</tr>
<tr>
<td>CCW Read 4 KB, flag 8 Off, 10 M</td>
<td></td>
</tr>
</tbody>
</table>

  In this case, the count is read into 20 MB and the 4 KB data is read into 10 M.
3.66 DASD extended format

You can allocate both sequential and VSAM data sets in extended format (EF) on a system-managed data set. The DASD volume must be attached to a controller that supports extended format. All the modern DASD controllers support this format.

It is recommended that you convert your data sets to extended format for better performance, additional function, and improved reliability. For example, there are several VSAM functions only available for EF data sets, such as:

- Data striping
- Data compression
- VSAM extended addressability
- Partial space release
- System-managed buffering

Extended format is a technique that affects the way count key data (CKD) is stored in a 3390 DASD track. It improves the performance and the reliability of an I/O operation.
The major difference between these data sets and data sets that are not in extended format is explained here:

- A 32-byte suffix is added to each physical record at the data component. This physical block suffix may increase the amount of space actually needed for the data set, depending on the blksize.

  The 32-byte suffix contains the following:
  - A relative record number used for data stripping.
  - A 3-byte field to detect controller invalid padding, thus improving the availability of the I/O operation. A discussion of this topic is beyond the scope of this book.

Record format on DASD
The physical record format and the suffix are transparent to the application; that is, the application does not require internal or external modifications to create and use the extended format data set.

Note: All channel programs accessing EF data sets must use data chaining to read the 32-byte suffix to a different area as a scatter-read or scatter-write I/O operation.

DFSMS SMS-managed data sets
All extended format data sets must be system-managed; that is, they have an associated storage class. To convert a non-extended format data set to an extended format, or to allocate an extended format data set, you need to create an SMS data class (DC) with the DATASETNAME TYPE field equal to EXT, and then assign the data sets to that data class.
3.67 Using MIDAWs

MIDAW facility
The MIDAW facility provides a more efficient CCW/IDAW structure for certain categories of data chaining I/O operations, as described here:

- MIDAW can significantly improve FICON performance for extended format data sets.
  - Non-extended data sets can also benefit from MIDAW.
- MIDAW can improve channel utilization and can significantly improve I/O response time.
  - It reduces FICON channel connect time, director ports and control unit overhead.

IDAWs
MIDAW means Modified IDAW. An IDAW is an Indirect Address Word that is used to specify real data addresses for I/O operations in a virtual environment. As discussed, the existing IDAW design allows the first IDAW in a list to point to any address within a page. Subsequent IDAWs in the same list must point to the first byte in a page. Also, all but the first and last IDAW in a list must deal with a complete 4 KB of data.

MIDAW format
The MIDAW facility is a new method of gathering/scattering data from and into discontinuous storage locations during an I/O operation. The modified IDAW (MIDAW) format is shown in Figure 3-68.

Figure 3-68  MIDAW CCW chaining

MIDAWs are a new method of gathering/scattering data into/from discontiguous storage locations during an I/O operation.
Each MIDAW has 128 bits. It is 16 bytes in length and is aligned on a quadword. Its most important new field is the count describing the length of the I/O buffer piece described in the MIDAW. With such a count, we save CCWs in the channel program, making it faster.

![Figure 3-69 Modified IDAW (MIDAW) format](image)

**Figure 3-69 Modified IDAW (MIDAW) format**

**MIDAW CCW chaining**

The use of MIDAWs is indicated by the MIDAW bit 7 in the CCW flags. The data count in the CCW should equal the sum of the data counts in the MIDAWs. The CCW operation ends when the CCW count goes to zero (0) or the last MIDAW (with the last flag) ends.

The combination of the address and count in a MIDAW cannot cross a page boundary; this means the largest possible count is 4 K. The maximum data count of all the MIDAWs in a list cannot exceed 64 K. (This is because the associated CCW count cannot exceed 64 K.)

The scatter-read or scatter-write effect of the MIDAWs makes it possible to efficiently send small control blocks (as the count and the EF suffix) embedded in a physical record to separate buffers than those used for larger data areas within the record. MIDAW operations are on a single I/O block, in the manner of data chaining. (Do not confuse this operation with CCW command chaining.)

**Extended format data sets**

z/OS extended format data sets use internal structures (usually not visible to the application program) that require a scatter-read (or scatter-write) operation. This means that CCW data chaining is required, and this produces less than optimal I/O performance. Extended format data sets were introduced in 1993 and are widely in use today. The most significant performance benefit of MIDAWs is achieved with extended format (EF) data sets.

To process an I/O operation to an EF data set would normally require at least two CCWs with data chaining. One CCW would be used for the 32-byte suffix of the EF data set. With MIDAW, the additional CCW for the EF data set suffix can be eliminated.

MIDAWs benefit both EF and non-EF data sets. For example, to read twelve 4 K records from a non-EF data set on a 3390 track, Media Manager (VSAM I/O driver) would chain 12 CCWs together using data chaining. To read twelve 4 K records from an EF data set, 24 CCWs would be chained (2 CCWs per 4K record). Now, by using Media Manager track-level command operations and MIDAWs, a whole track can be transferred using a single CCW.

**Performance benefits**

Media Manager has the I/O channel programs support for implementing EF data sets; it automatically exploits MIDAWs when appropriate. Today, most disk I/Os channel programs in the system are generated using Media Manager.
Reducing CCWs using MIDAW

By using MIDAW, Media Manager can transfer a whole track using a single CCW, without the need of data chaining, as shown in Figure 3-70.

The modified indirect data address word (MIDAW) facility is a system architecture and software exploitation designed to improve FICON performance. This facility is only available on System z9 and IBM System z10 Enterprise Class servers, and is exploited by the Media Manager in z/OS.

MIDAWs benefit both EF and non-EF data sets. For example, to read twelve 4 K records from a non-EF data set on a 3390 track, Media Manager would chain 12 CCWs together using data chaining. To read twelve 4 K records from an EF data set, 24 CCWs would be chained (two CCWs per 4 K record). Using Media Manager track-level command operations and MIDAWs, an entire track can be transferred using a single CCW.

Performance benefits

z/OS Media Manager has the I/O channel programs support for implementing EF data sets and it automatically exploits MIDAWs when appropriate. Today, most disk I/Os in the system are generated using media manager. Media Manager with the MIDAW facility can provide significant performance benefits when used in combination applications that use EF data sets (such as DB2) or long chains of small blocks.
3.69 MIDAW performance results

MIDAW exploiting

MIDAWs are used by two IOS drivers:
- The Media Manager (a VSAM-specific I/O driver) exploits MIDAWs when appropriate.
- Users of the EXCPVR IOS driver may construct channel programs containing MIDAWs, provided that they construct an IOBE with the new IOBEMIDA bit set.

Note: Users of the EXCP driver may not construct channel programs containing MIDAWs.

MIDAW performance results

Media Manager contains the I/O channel program support for implementing Extended Format data sets, and it automatically exploits MIDAWs when appropriate. Today, most disk I/Os in the system are generated using Media Manager.

The MIDAW facility removes the 4 K boundary restrictions of IDAWs, and in the case of EF data sets, reduces the number of CCWs. Decreasing the number of CCWs helps to reduce the FICON channel utilization. Media Manager and MIDAWs will not cause the bits to move any faster across the FICON link, but they do reduce the number of frames and sequences flowing across the link, thus utilizing the channel resources more efficiently.

Use of the MIDAW facility with FICON Express4, operating at 4 Gbps, compared to use of Indirect Data Address Words (IDAWs) with FICON Express2, operating at 2 Gbps, showed an improvement in throughput of greater than 220% for all reads (270 MBps versus 84 MBps) on DB2 table scan tests with extended format data sets.
These measurements are examples of what has been achieved in a laboratory environment using one FICON Express4 channel operating at 4 Gbps (CHPID type FC) on a z9 EC with z/OS V1.7 and DB2 UDB for z/OS V8.

The performance of a specific workload may vary, according to the conditions and hardware configuration of the environment. IBM laboratory tests found that DB2 gains significant performance benefits using the MIDAW facility in the following areas:

- Table scans
- Logging
- Utilities
- Using DFSMS striping for DB2 data sets

Figure 3-71 on page 273 illustrates the environment where the results were captured. By all metrics, we observe a dramatic improvement in the I/O performance, as listed here:

- I/O connect time: more than 50% decrease, for three DB2 partitions
- Channel busy%: 30% decrease, for three DB2 partitions
- Throughput in MB/sec: 63% increase, for three DB2 partitions
3.70 Cryptography concepts

To encrypt data means that by applying an encrypt algorithm to a secret key and data in clear, you produce encrypted data. To decrypt data in clear means applying the same encrypt algorithm to a secret key and encrypted data produces back the data in clear. Refer to Figure 3-72.

The algorithm is named symmetric when keys used for encrypt and decrypt are identical.

The algorithm is named asymmetric when the keys used for encrypt and decrypt are different. In this case they are called private keys and public keys.

A large amount of processor power is needed to implement the majority of the cryptography algorithms, mainly because of the current size of used keys. In order to address this issue, the crypto functions are usually off-loaded to hardware coprocessors, as in z196 CPC.

The algorithm is named synchronous when the main processor loops waiting for the crypto coprocessor to execute the requested crypto function.

The algorithm is named asynchronous when the main processor does not loop waiting for the crypto coprocessor to execute the requested crypto function. The dispatchable unit (TCB or SRB, in z/OS) requesting the crypto function is placed in a wait state by the operating system. In this case a wake-up call mechanism needs to be implemented.
Usually, the asynchronous algorithm takes a longer time to be executed (when compared with synchronous), due to the distance of the outbound coprocessor from the processor.

Processor time is a measure of the time that a job or task controls the processor. MVS records the SRB and TCB times as service units. MVS converts each of the times into service units by multiplying the time by an SRM processor-dependent constant. For example, MVS converts the time a job executes under a TCB into TCB service units by multiplying the TCB time by an SRM constant that is processor-dependent. A processor’s consumed time is a measure of the dispatchable unit (TCB/SRB).

Today, e-business and e-commerce applications are increasingly relying on cryptographic techniques to provide the confidentiality and authentication required in this environment. For example, Secure Sockets Layer/Transport Layer Security (SSL/TLS) technology is a key technology for conducting secure e-commerce using web servers, and it is in use by an increasing number of e-business applications, demanding new levels of security and performance.
3.71 Cryptography in z196

- Crypto enablement feature (CPACF)
- Crypto Express2
- TKE workstation
  - TKE smart card reader

Figure 3-73  Cryptographic z196 hardware features

Cryptography in z196
IBM has a long history of providing hardware cryptographic solutions, from the development of Data Encryption Standard (DES) in the 1970s to delivering integrated cryptographic hardware in a server to achieve the US Government's highest security rating FIPS 140-2 Level 4 for secure cryptographic hardware.

z196 cryptographic features
Three types of hardware cryptographic features are available on the z196:
- CP Assist Crypto Function (CPACF) in the coprocessor at the PU chip.
- Optional Crypto Express-3 PCIe processors
- Optional TKE workstation

Note: The cryptographic features are usable only when explicitly enabled through IBM to conform with US export requirements.

All the hardware z196 cryptographic facilities are exploited by the z/OS component Integrated Cryptographic Service Facility (ICSF) and the IBM Resource Access Control Facility (RACF), or equivalent software products. They provide data privacy, data integrity, cryptographic key installation and generation, electronic cryptographic key distribution, and personal identification number (PIN) processing.
CP assist crypto function (CPACF)

This is a microcode assist function plus hardware component implemented in a coprocessor (shared with a data compress function) accessed by a pair of PUs within the PU chip in an MCM. Refer to “PU chip coprocessor” on page 193. CPACF provides high performance encryption and decryption support. It is a hardware-synchronous implementation; that is, holding processor processing of the instruction flow until the operation completes. Several instructions are able to invoke the CPACF, when executed by the PU:

- KMAC Compute Message Authentic Code
- KM Cipher Message
- KMC Cipher Message with Chaining
- KMF Cipher Message with CFB
- KMCTR Cipher Message with Counter
- KMO Cipher Message with OFB
- KIMD Compute Intermediate Message Digest
- KLMD Compute Last Message Digest
- PCKMO Provide Cryptographic Key Management Operation

CPACF offers a set of symmetric (meaning that encryption and decryption processes use the same key) cryptographic functions that enhance the encryption and decryption performance of clear key operations or SSL, VPN, and data storing applications that do not require FIPS 140-2 level 4 security. Clear key means that the key used is located in central storage.

These functions are directly available to application programs, thereby diminishing programming overhead of going through ICSF. The CPACF complements, but does not execute, public key (PKA) functions. Note that keys, when needed, are to be provided in clear form only.

Traditionally CPACF was only able to execute clear-key crypto algorithms. This has changed. Refer to “Protected keys in CPACF” on page 284.

Crypto express-3

Each Crypto Express-3 feature contains two PCI-X adapters. There are up to eight such features in a z196. Each adapter can be configured as a cryptographic coprocessor (as the former PCICC) or accelerator (as the former PCICA). During the feature installation, both PCI-X adapters are configured by default as coprocessors. The Crypto Express-3 feature does not use CHPID from the channel subsystem pool, but each feature is assigned two PCHIDs, one per PCI-X adapter. For more information about this topic, refer to “Crypto express-3” on page 280.

TKE workstation

The TKE workstation is an IBM PCI bus-based personal computer. It is used to enter a non-clear key (never stored in memory) into the Crypto Express-3 hardware. You can use smart cards as well. Refer to “Protected keys in CPACF” on page 284.
3.72 z196 crypto synchronous functions

- Data encryption/decryption algorithms
  - Data Encryption Standard (DES)
    - Double length-key DES
    - Triple length-key DES
  - Advanced Encryption Standard (AES) for 128-bit keys

- Hashing algorithms: SHA-1 and SHA-256

- Message authentication code (MAC):
  - Single-key MAC and
  - Double-key MAC

- Pseudo random number generation (PRNG)

Figure 3-74 Synchronous crypto functions in z196

Cryptographic synchronous functions

Cryptographic synchronous functions are provided by CPACF. *Synchronous* means that the function holds processor processing until the crypto operation has completed. Note that crypto keys, when needed, are to be provided in *clear* form only, meaning that the key may be in central storage.

The following algorithms are available:

- Data encryption and decryption algorithms
  - Data Encryption Standard (DES)
    - Double-length key DES
    - Triple-length key DES (TDES)
  - Advanced Encryption Standard (AES) for 128-bit, 192-bit, and 256-bit keys
- Hashing algorithms: SHA-1, SHA-256, SHA-384, and SHA-512
- Message authentication code (MAC):
  - Single-key MAC
  - Double-key MAC
- Pseudo Random Number Generation (PRNG) and Random Number Generation (RNG) with 4096-bit RSA support

The CPACF functions are supported by z/OS, z/VM, z/VSE, and Linux on System z.
3.73 Crypto express-3

The Crypto Express-3 feature is an outboard coprocessor. It is peripheral, being located in the I/O cage in one I/O card. To make it simple, it looks like a channel coupled to a crypto coprocessor. All its processing is asynchronous because the processor does not wait for the crypto request completion. The end of a request is indicated by turning on a bit in the HSA vector area. These bits are inspected by the z/OS dispatcher. Crypto Express-3 provides a high-performance cryptographic environment with added functions.

Crypto Express3 represents the newest-generation cryptographic feature designed to complement the cryptographic functions of CPACF. The Crypto Express3 resides in the I/O cage of the z196 (or z10 EC) or in the I/O drawer of z196 (or z10 BC), and continues to support all of the cryptographic functions available on Crypto Express2.

Crypto Express3 is a state-of-the-art, tamper-sensing and tamper-responding, programmable cryptographic feature. The cryptographic electronics and microprocessor provide a secure cryptographic environment using two PCI-Express (PCI-E) adapters. Each PCIe adapter contains dual processors that operate in parallel to support the IBM Common Cryptographic Architecture (CCA) with high reliability.

Each feature has two Peripheral Component Interconnect eXtended (PCI-X) cryptographic adapters. Each PCI-X cryptographic adapter can be configured as a cryptographic coprocessor or a cryptographic accelerator.
PPCI-X cryptographic adapters, when configured as coprocessors, are designed for FIPS 140-2 Level 4 compliance rating for secure cryptographic hardware modules. Note that unauthorized removal of the adapter or feature zeroes its content.

The Crypto Express-3 in z196 replaces PCIXCC and PCICA. The reconfiguration of the PCI-X cryptographic adapter between coprocessor and accelerator mode is exclusive to z196, z10 EC and z9 servers, and is also supported for Crypto Express-3 features brought forward from z990 and z890 systems to the z9 and z10 EC.

- When the PCI-X cryptographic adapter is configured as a coprocessor, the adapter provides equivalent functions (plus some additional functions) to the PCICC card on previous systems with a doubled throughput.
- When the PCI-X cryptographic adapter is configured as an accelerator, it provides PCICA-equivalent functions with an expected throughput of approximately three times the PCICA throughput on previous systems.

Then, each Crypto Express3 feature may be configured as:

- Two PCI-E cryptographic coprocessors (default mode)
- One PCI-E cryptographic coprocessor and one PCI-E cryptographic accelerator
- Two PCI-E cryptographic accelerators

The z196 supports up to eight Crypto Express-3 features (up to sixteen PCI-X cryptographic adapters) to be installed.

The security-relevant portion of the cryptographic functions is performed inside the secure physical boundary of a tamper-resistant card. Master keys and other security-relevant information is also maintained inside this secure boundary.

It does not use CHPID, but requires one slot in the I/O cage and one PCHID for each PCI-X cryptographic adapter. The feature is attached to an IFB-MP and has no other external interfaces. Each PCI-X cryptographic adapter can be shared by any logical partition defined in the system, up to a maximum of 16 logical partitions per PCI-X cryptographic adapter.

The Crypto Express-3 coprocessor enables the user to:

- Encrypt and decrypt data utilizing secret-key (non-clear key) algorithms. Triple-length key DES and double-length key DES algorithms are supported.
- Generate, install, and distribute cryptographic keys securely, using both public and secret key cryptographic methods.
- Generate, verify, and translate personal identification numbers (PINs).
- Ensure the integrity of data by using message authentication codes (MACs), hashing algorithms, and Rivest-Shamir-Adelman (RSA) public key algorithm (PKA) digital signatures.

**Note:** While PCI-X cryptographic adapters have no CHPID type and are not identified as external channels, all LPs in all LCSSs have access to the coprocessor (up to 16 LPs per coprocessor).
3.74 z196 crypto asynchronous functions

- Data encryption/decryption algorithms
  - Data Encryption Standard (DES)
  - Double length-key DES
  - Triple length-key DES

- DES key generation and distribution

- PIN generation, verification and translation functions

- Public Key Security Control (PKSC)

- Public Key Algorithm (PKA) Facility

---

**z196 crypto asynchronous functions**

These functions are provided by the PCI-X cryptographic adapters.

The following secure key (not clear key) functions are provided as cryptographic asynchronous functions. System internal messages are passed to the cryptographic coprocessors to initiate the operation, and messages are passed back from the coprocessors to signal completion of the operation.

- Data encryption and decryption algorithms
  - Data Encryption Standard (DES)
  - Double length-key DES
  - Triple length-key DES
- DES key generation and distribution
- PIN generation, verification, and translation functions
- Pseudo random number generator (PRNG)
- Public Key Algorithm (PKA) Facility
  - Importing RSA public-private key pairs in clear and encrypted forms
  - Rivest-Shamir-Adelman (RSA)
    - Key generation, up to 2048-bit
    - Signature verification, up to 2048-bit
- Import and export of DES keys under an RSA key, up to 2048-bit
  - Public Key Encrypt (PKE)
    Public Key Encrypt service is provided for assisting the SSL/TLS handshake. When used with the Mod_Raised_to Power (MRP) function, it is also used to offload compute-intensive portions of the Diffie-Hellman protocol onto the PCI-X cryptographic adapter.
  - Public Key Decrypt (PKD)
    Public Key Decrypt supports a Zero-Pad option for clear RSA private keys. PKD is used as an accelerator for raw RSA private operations such as those required by the SSL/TLS handshake and digital signature generation. The Zero-Pad option is exploited by Linux to allow use of a PCI-X cryptographic adapter for improved performance of digital signature generation.
  - Derived Unique Key Per Transaction (DUKPT)
    The service is provided to write applications that implement the DUKPT algorithms as defined by the ANSI X9.24 standard. DUKPT provides additional security for point-of-sale transactions that are standard in the retail industry. DUKPT algorithms are supported on the Crypto Express-3 feature coprocessor for triple-DES with double-length keys.
  - Europay Mastercard Visa (EMV) 2000 standard
    Applications may be written to comply with the EMV 2000 standard for financial transactions between heterogeneous hardware and software. Support for EMV 2000 applies only to the Crypto Express-3 feature coprocessor of the z196.

Other key functions of the Crypto Express-3 feature serve to enhance the security of public/private key encryption processing:
- Retained key support (RSA private keys generated and kept stored within the secure hardware boundary)
- Support for 4753 Network Security Processor migration
- User-Defined Extensions (UDX) support, including:
  - For Activate UDX requests:
    - Establish Owner
    - Relinquish Owner
    - Emergency Burn of Segment
    - Remote Burn of Segment
  - Import UDX File function
  - Reset UDX to IBM default function
  - Query UDX Level function

UDX allows the user to add customized operations to a cryptographic coprocessor. User-Defined Extensions to the Common Cryptographic Architecture (CCA) support customized operations that execute within the Crypto Express-3 feature. UDX is supported via an IBM, or approved third-party, service offering.
3.75 Protected keys in CPACF

Since PCIXCC deployment (old zSeries machines), secure keys are processed on the PCI-X and PCIe cards, requiring an asynchronous operation to move the data and keys from the general purpose CP to the crypto cards. Clear keys (non-secure) process faster than protected keys because the process is done synchronously on the CPACF. Protected keys blend the security of Crypto Express3 (CEX3) and the performance characteristics of the CPACF, running closer to the speed of clear keys.

An enhancement to CPACF facilitates the continued privacy of cryptographic key material when used for data encryption. In Crypto Express3, a secure key is encrypted under a master key, whereas a protected key is encrypted under a wrapping key that is unique to each LPAR.

When the wrapping key is unique to each LPAR, a protected key cannot be shared with another LPAR. CPACF, using key wrapping, ensures that key material is not visible to applications or operating systems during encryption operations.

CPACF code generates the wrapping key and stores it in the protected area of hardware system area (HSA). The wrapping key is accessible only by firmware. It cannot be accessed by operating systems or applications. DES/T-DES and AES algorithms were implemented in CPACF code with support of hardware assist functions. Two variations of wrapping key are generated, one for DES/T-DES keys and another for AES keys.
Wrapping keys are generated during the clear reset each time an LPAR is activated or reset. There is no customizable option available at SE or HMC that permits or avoids the wrapping key generation. Figure 3-77 on page 284 shows this function.

If a CEX3 coprocessor is available, a protected key can begin its life as a secure key. Otherwise, an application is responsible for creating or loading a clear key value and then using the new PCKMO instruction to wrap the key. ICSF is not called by application if CEX3C is not available.

A new segment in profiles at the CSFKEYS class in RACF restricts which secure keys can be used as protected keys. By default, all secure keys are considered not eligible to be used as protected keys. The process described in Figure 3-77 on page 284 considers a secure key as being the source of a protected key.

In Figure 3-77 on page 284, the source key is already stored in CKDS as a secure key (encrypted under the master key). This secure key is sent to CEX3C to be deciphered and sent to CPACF in clear text. At CPACF, the key is wrapped under the LPAR wrapping key and then it is returned to ICSF. After the key is wrapped, ICSF can keep the protected value in memory, passing it to the CPACF, where the key will be unwrapped for each encryption/decryption operation. The protected key is designed to provide substantial throughput improvements for a large volume of data encryption as well as low latency for encryption of small blocks of data. A high performance secure key solution, also known as a protected key solution, requires ICSF HCR7770, and it is highly desirable to use a Crypto Express3 card.
3.76 PR/SM and cryptography

Figure 3-78  PR/SM and cryptography

PR/SM and cryptography
All logical partitions (LPs) in all Logical Channel Subsystems (LCSS) have access to the Crypto Express-3 feature (two ports), up to 32 LPs per feature.

PR/SM fully supports the Crypto Express-3 feature coprocessor to establish a logically partitioned environment in which multiple LPs can use and share the cryptographic functions. There is one master key per each LP. Several sizes of data protection master key, and one Public Key Algorithm (PKA) master key are provided for each of 16 cryptographic domains that a coprocessor can serve. Each cryptographic coprocessor has 16 physical sets of registers or queue registers, each set belonging to a domain, as follows:

- A cryptographic domain index, from 0 to 15, is allocated to an LP via the definition of the partition in its image profile. The same domain must also be allocated to the ICSF instance running in the LP via the Options Data Set.

- Each ICSF instance accesses only the Master Keys or queue registers corresponding to the domain number specified in the LP image profile at the Support Element and in its Options Data Set. Each ICSF instance is seeing a logical crypto coprocessor consisting of the physical cryptographic engine and the unique set of registers (the domain) allocated to this LP.
**Note:** Cryptographic coprocessors are not tied to logical partition numbers or MIF IDs. They are set up with PCI-X adapter numbers and domain indices that are defined in the partition image profile. The customer can assign them to the partition and change or clear them when needed.
3.77 Just-in-time concurrent upgrades

Given today's business environment, benefits of the concurrent capacity growth capabilities provided by z199 CPC are plentiful, and include:

- Enabling exploitation of new business opportunities
- Supporting the growth of e-business environments
- Managing the risk of volatile, high growth, high volume applications
- Supporting 24x7 application availability even in a disaster situation
- Enabling capacity growth during “lock down” periods.

Figure 3-79  Just-in-time concurrent upgrades

Just-in-time concurrent upgrades
Here we talk about non-disruptively activating (or inactivating) PUs and central storage that are already in the books, and I/O channels already in I/O slots, without the need of IBM customer engineer actions.

The z196 allows just-in-time concurrent (non-disruptive) upgrades, adding more capacity to the HW, without an outage in the delivered service.

Given today's business environment, benefits of the concurrent capacity growth capabilities provided by the z199 CPC are plentiful, and include:

- Enabling exploitation of new business opportunities
- Supporting the growth of e-business environments
- Managing the risk of volatile, high growth, high volume applications
- Supporting 24x7 application availability even in a disaster situation
- Enabling capacity growth during “lock down” periods

This capability is based on the flexibility of the z196 system design and structure, which allows configuration control by the Licensed Internal Code (LIC) and concurrent hardware installation.
**Licensed internal code (LIC)-based upgrades**

LIC-Configuration Control (LIC-CC) provides for server upgrade with no hardware changes by enabling the activation of additional previously installed capacity. Concurrent upgrades via LIC-CC can be done for:

- Processing units (CPs, IFLs, and ICFs) - require available spare PUs on installed books.
- Memory - requires available capacity on installed memory cards. Refer to “Purchase memory offerings” on page 222.
- I/O card ports (ESCON channels and ISC-3 links) - require available ports on installed I/O cards.

Hardware installation configuration upgrades can also be concurrent by installing additional:

- Books (which contain PUs, memory, and STIs) - requires available book slots in the installed server cage.
- I/O cards - requires available slots on installed I/O cages; I/O drawers can be installed concurrently and this does not apply to I/O cages.
3.78 On/Off capacity on demand (CoD)

On/Off CoD provides temporary capacity for all types of characterized PUs to serve an expected peak on demand.

In z196 the use of resource tokens may help the decrease of the hardware costs associated with using On/Off CoD:

- For CP capacity, a resource token represents an amount of processing capacity that will result in 1 MSU of SW cost for 1 day - an MSU-day.
- For specialty engines, a resource token represents activation of 1 engine of that type for 1 day – an IFL-day, a zIIP-day or a zAAP-day (specialty engine-day).

On/Off capacity on demand (CoD)

Here we start to cover the several IBL plans for capacity upgrades.

On/Off CoD provides temporary capacity for all types of characterized PUs to serve an expected peak on PU demand. Spare PUs that are currently unassigned and unowned can be temporarily and concurrently activated as any characterizable PU via LIC-CC, up to the limits of the physical CPC size.

This means that an On/Off CoD upgrade cannot change the z196 CPC model, as additional book(s) installation is not supported. However, On/Off CoD changes the CPC's software number (7XX). When you request temporary capacity, consider the following guidelines:

- Temporary capacity must be greater than permanent capacity.
- Temporary capacity cannot be more than double the purchased capacity.
- On/Off CoD cannot decrease the number of engines on the server.
- Adding more engines than are currently installed is not possible. Helps bound the hardware costs associated with using On/Off CoD.

On/Off CoD can be exploited by the WLM capacity provisioning function. Refer to “Capacity provisioning” on page 294, for more information.

To better exploit the On/Off CoD function, an initial configuration should be carefully planned to allow a concurrent upgrade up to a target configuration. You need to consider planning,
positioning, and other issues to allow an On/Off CoD nondisruptive upgrade. By planning ahead, it is possible to enable nondisruptive capacity for the z196, without system power-down and no associated POR or IPLs.

**Limiting On/Off CoD capacity through resource tokens**

This function allows Purchase Order (PO) customers to assign tokens to limit their financial obligation.

- For CP capacity, a resource token represents an amount of processing capacity that will result in 1 MSU of SW cost for 1 day - an MSU-day.
- For specialty engines, a resource token represents activation of 1 engine of that type for 1 day – an IFL-day, a zIIP-day or a zAAP-day (specialty engine-day).

Tokens are decremented at the end of each 24-hour period. If for any engine type there are insufficient tokens for the next 24-hour period, the entire record is deactivated. Customers are notified when the token pool falls below five days remaining (again at 24 hours). Customers can order replenishment records to add tokens.

There are two optional Prepaid or Post-paid plans:

- When prepaid, you are billed for the total amount of resource tokens contained within the On/Off CoD record when the record is downloaded.
- When post-paid, the total billing against the On/Off CoD record is limited by the total amount of resource tokens contained within the record.
3.79 Other capacity upgrade plans

Other capacity upgrade plans:

- Customer Initiated Upgrade (CIU) provides replacement backup capacity (PUs and memory) for planned downtime events, via the web IBM Resource Link
- Capacity BackUp (CBU) provides reserved emergency PUs backup capacity for unplanned situations where customers have lost capacity in another site
- Capacity for Planned Events (CPE) provides PUs replacement backup capacity for planned downtime events

Customer Initiated Upgrade (CIU)

CIU is the capability for the z196 user to initiate planned (permanent or temporary) upgrades for PUs and/or memory via the web, using the IBM Resource Link.

CIU is similar to CUoD, but the capacity growth can be added by the customer. The customer also has the ability to unassign previously purchased PUs. The customer will then be able to download and apply the upgrade using functions on the HMC via the Remote Support Facility, without requiring the assistance of IBM service personnel.

After all the prerequisites are in place, the whole process, from ordering to activation of the upgrade, is performed by the customer. The actual upgrade process is fully automated and does not require any on-site presence by IBM service personnel.

CIU supports LIC-CC upgrades only, and does not support I/O upgrades. All additional capacity required by a CIU upgrade must be previously installed. This means that additional books and/or I/O cards cannot be installed via CIU. CIU may change the server’s software model (7XX) but cannot change the z9 EC server model.

Before customers are able to use the CIU function, they have to be registered. After they are registered, they gain access to the CIU application by ordering the CIU Registration feature from their salesperson.
Capacity BackUp (CBU)
CBU is offered to provide reserved emergency PUs backup capacity for unplanned situations where customers have lost capacity in another part of their establishment, and want to recover by adding the reserved capacity on a designated z196 CPC.

CBU is a quick, temporary activation of processor units (CP, zAAP, zIIP, IFL, ICF, SAP) in the face of a loss of processing capacity due to an emergency or disaster/recovery situation.

Note: CBU is for disaster/recovery purposes only, and cannot be used for peak load management of customer workload.

Capacity for Planned Events (CPE)
A CPE plan is offered with the z196 to provide replacement backup capacity for planned downtime events. For example, if a customer needs to perform extension or repair work in a server room, replacement capacity can be installed temporarily on another z196 in the customer's environment CPE.

CPE is intended to replace capacity lost within the enterprise due to a planned event such as a facility upgrade or system relocation. CPE is intended for short duration events lasting up to a maximum of three days. Each CPE record, after it is activated, gives the customer access to all dormant PUs on the server. Processing units can be configured in any combination of CP or specialty engine types (zIIP, zAAP, SAP, IFL, ICF). The capacity needed for a given situation is determined by the customer at the time of CPE activation.

The processors that can be activated by CPE come from the available spare PUs on any installed book. CPE features can be added to an existing z196 nondisruptively. There is a one-time fixed fee for each individual CPE event. The base server configuration must have sufficient memory and channels to accommodate the potential needs of the large CPE-configured server. It is important to ensure that all required functions and resources are available on the server where CPE is activated, including CF LEVELs for Coupling Facility partitions, memory, and cryptographic functions, as well as connectivity capabilities.
3.80 Capacity provisioning

**z/OS Capacity Provisioning**

Here we explain that instead of activating and deactivating PUs manually through On/Off CoD, it can be done autonomically through a z/OS Workload Manager (WLM) function named Capacity Provisioning.

As we know, WLM manages the workload by goals and business importance (as defined by an installation WLM policy) on each z/OS system in a Parallel Sysplex. WLM metrics (such as resource delays and performance index) are available through existing interfaces and are reported through RMF Monitor III, with one RMF data gatherer per z/OS system.

Prior to z/OS 1.9, WLM controlled, through the IRD feature, the distribution of physical CPUs among the z/OS systems contained in an LPAR cluster. This task was executed by the following WLM functions:

- WLM Vary Logical CPU Management, where the number of logical CPUs in an LP is dynamically controlled by WLM.
- WLM Vary Weight Management, where depending whether goals are being achieved or not, the LP weights are dynamically modified taking processor resource from an LP and delivering to another in the same server.

With the z196 provisioning capability combined with the Capacity Provisioning Management (CPM) WLM component in z/OS, it is possible in a new, flexible and automated process to control the activation and deactivation of PUs through the On/Off CoD.
Also, previously WLM was only able to take processor capacity from an LP and give it to another LP to decrease the processor delays causing important unhappy transactions. Now, on top of that, CPM is able to activate spare processors through On/Off CoD to decrease those processor delays.

The z/OS provisioning environment is shown in Figure 3-82 on page 294, along with all of its components.

It works like this:

- RMF has a distributed data server (DDS) facility where all data captured by all the RMF in a sysplex can be sent to an RMF focal point through the network.
- The RMF Common Interface Module (CIM) providers and associated CIM models publish the RMF Monitor III data.

CIM is a standard data model developed by a consortium of major HW/SW vendors (including IBM) known as the Distributed Management Task Force (DMTF). It is a part of the Web Based Enterprise Management (WBEM) initiative. It includes a set of standards and technologies that provide management solutions for distributed network environments. CIM creates an interface (API) where applications, for example, can ask system management questions such as, how many jobs are running in the system? This query must be converted to an API known by the running operating system. In z/OS, CIM is implemented through the Common Event Adapter (CEA) address space.

- Then the Capacity Provisioning Manager (CPM), a function inside z/OS, retrieves critical metrics from one or more z/OS systems through the Common Information Model (CIM) structures and protocol. Depending on such metrics, CPM communicates to (local or remote) support elements and HMC, respectively, via the SNMP protocol to access On/Off Capacity on Demand. Refer to “SNMP interface to HMC” on page 298. The control over the Provisioning Infrastructure is executed by the CPM through the Capacity Provisioning Policy (CPP) that controls the Capacity Provisioning Domain (CPD). Refer to “Capacity provisioning domain” on page 296 for more information about a CPD.

- The Capacity Provisioning Policy (CPP) is created and managed by the Capacity Provisioning Control Center (CPCC), which resides on a workstation providing a system programmer front-end to administer such policies. CPCC is a Graphical User Interface (GUI) component. The CPCC is installed on a Microsoft Windows workstation. These policies are not the ones managed by WLM and kept in the WLM couple data set. The CPCC is not required for regular CPM operation due to defaults. Refer to “SNMP interface to HMC” on page 298, to learn more about CPP.
3.81 Capacity provisioning domain

Capacity provisioning domain (CPD) represents the set of MVSs that are controlled by CPM, the Capacity Provisioning Manager. The HMC of the CPCs within a CPD must be connected to the same processor HMC network. Parallel Sysplex members can be part of a CPD. There is no requirement that all z/OS members of a Parallel Sysplex must be part of the CPD, but participating z/OS members must all be part of the same CPD.

Administrators work through the CPCC interface to define domain configurations and provisioning policies, but it is not needed during production.

CPM operates in four different modes allowing for different levels of automation:

- Manual mode
  - Command driven mode - no CPM policy active
- Analysis mode
  - CPM processes capacity provisioning policies and informs the operator when a provisioning or de-provisioning action would be required according to policy criteria.
  - It is up to the operator whether to ignore the information or to manually upgrade or downgrade the system using either the HMC, the SE, or available CPM commands.
▶ Confirmation mode
  – CPM processes capacity provisioning policies and interrogates the installed temporary offering records. Every action proposed by the CPM needs to be confirmed by the operator.
▶ Autonomic mode
  – This mode is similar to the confirmation mode, but no operator confirmation is needed.
3.82 SNMP interface to HMC

![Diagram of SNMP interface with HMC]

**Capacity provisioning policy (CPP)**

The provisioning policy defines the circumstances under which additional capacity may be provisioned. There are three elements in the criteria:

- **When** provisioning is allowed (time condition):
  - Start time - indicates when provisioning can begin.
  - Deadline - provisioning of additional capacity no longer allowed.
  - End time - deactivation of additional capacity should begin.

- **Which** work qualifies for provisioning - parameters include (workload condition):
  - The z/OS systems that may execute eligible work (domain definition)
  - Importance filter - eligible service class periods, identified by WLM importance
  - Performance Indicator (PI) criteria:
    - Activation threshold - PI of service class periods must exceed the activation threshold for a specified duration before the work is considered to be suffering.
    - Deactivation threshold - PI of service class periods must fall below the deactivation threshold for a specified duration before the work is considered to be no longer suffering.
  - Included Service Classes - eligible service class periods
  - Excluded Service Classes - service class periods that should not be considered
How much additional capacity may be activated, or the provisioning scope. Specified in MSUs (per processor), number of zAAPs, and number of zIIPs; one specification per CPC that has MVS, as a part of the Capacity Provisioning Domain.

SNMP interface with HMC
In Figure 3-84 on page 298, we describe the interface used by CPM through the Support Element interface in order to activate or inactivate PUs. The request from CPM for processors is in terms of MSUs. The Support Element response may be in upgrading a sub-uni (for example, a 605 to 705 or adding more 7xx). You may see that GDPS is also a user of such interfaces.
Chapter 4. zEnterprise BladeCenter Extension Model 002 (zBX)

The zBX is a new System z infrastructure built to house our high performance optimizers and select IBM POWER7® and System x® general purpose blades. Blades will allow us to expand our application portfolio with more AIX and Linux applications.

The zBX brings computing capacity of systems in blade form-factor to the zEnterprise System. It is designed to provide a redundant hardware infrastructure that supports the multiplatform environment of the zEnterprise System in a seamless integrated way.

IBM continues to invest in leveraging a large portfolio of z/OS and Linux on System z applications. It is important to note that zEnterprise is *not* about moving applications off of the mainframe. It is exactly the opposite.

The WebSphere Data appliance (SODed for 1H11) can help simplify, govern, and enhance the security of XML and IT services by providing connectivity, gateway functions, data transformation, protocol bridging, and intelligent load distribution.

Connectivity between the z196 and the zBX is with dedicated high performance private networks, one for data and one for support.

Complimentarily, zBX is a cheap solution to run UNIX applications from software houses where there is no support for running such at zLinux.

In this chapter we introduce the zBX Model 002 and describe its hardware components. We also explain the basic concepts and building blocks for zBX connectivity.

The information in this chapter can be used for learning purposes and to help define the configurations that best fit your requirements.
4.0.1 zEnterprise

The IBM zEnterprise System represents a new height for mainframe functionality and qualities of service. It has been rightly portrayed as a cornerstone for the IT infrastructure, especially when flexibility for rapidly changing environments is called for.

IBM zEnterprise System characteristics make it especially valuable for mission critical workloads. Today, most of these applications have multilayered architectures that span various hardware and software platforms. However, there are differences in the qualities of service offered by the platforms. There are also various configuration procedures for their hardware and software, operational management, software servicing, failure detection and correction, and so on. These in turn require personnel with distinct skill sets, various sets of operational procedures, and an integration effort that is not trivial and, therefore, not often achieved.

Failure in achieving integration translates to lack of flexibility and agility, which can impact the bottom line.

IBM mainframe systems have been providing specialized hardware and dedicated computing capabilities for a long time. This was often done by offloading many functions from PUs to coprocessors or microcode assists, such as:

- Vector facility of the IBM 3090
- System Assist Processor (SAP) for I/O handling; implemented the 370-XA architecture
- Coupling Facility
- Cryptographic processors
All the I/O cards are specialized dedicated hardware components, with sophisticated software such as FICON, CF links.

The common theme with all of these specialized hardware components is their seamless integration within the mainframe.

The zBX components are also configured, managed, and serviced the same way as the other components of the System z server. Despite the fact that the zBX processors might not be System z PUs, the zBX is in fact, handled by System z management firmware called the IBM zEnterprise Unified Resource Manager. The zBX hardware features are part of the mainframe, not add-ons.

System z has long been an integrated heterogeneous platform. As examples we have zLinux and Unix System Services (a z/OS component). With zBX, that integration reaches a new level. zBX provides within the System zEnterprise infrastructure a solution for running AIX workloads with IBM POWER7 (3 GHz) based blades. Also, zBX provides an optimized solution for running Data Warehouse and Business Intelligence queries against DB2 for z/OS, with fast and predictable response times, while retaining the data integrity, data management, security, availability, and other qualities of service of System z.

Figure 4-1 on page 302 shows the zBX made of four frames, respectively B, C, D and E.
4.1 zBX hardware rack components

The zBX (machine type of 2458-002) can be ordered with a new z196 or as an MES to an existing z196. Either way, the zBX is treated as an extension to a z196 and cannot be ordered as a standalone feature.

It is capable of hosting integrated multiplatform systems and heterogeneous workloads, with integrated advanced virtualization management in a combination of up to 112 different blades. The management support of a zBX is through the z196 Hardware Management Console (HMC) and Support Element (SE). zBX is attached with redundant OSA-Express3 10Gb Ethernet and 1000BASE-T to z196.

zBX has a redundant infrastructure for fault tolerance and higher availability.

There are one to four standard 19-inch 42U (industry-standard 19") high zEnterprise racks (frames) with required network and power infrastructure. Figure 4-3 on page 306 shows a z196 with a maximum zBX configuration. The first rack (Rack B) is the primary rack where the four top of rack (TOR) switches reside. The other three racks (C, D, and E) are expansion racks.

Each rack may have:

- Up to two BladeCenter chassis. Each BladeCenter chassis may have up to 14 blades.
Two top of rack (TOR) 1000BASE-T switches (Rack B only) for the intra node management network (INMN). A zBX can only be managed by one z196 (HMC and SE) through the INMN connections.

Two TOR 10 GbE switches (Rack B only) for the intra-ensemble data network (IEDN)

Power and Distribution Units and optional heat exchangers. Power and cooling are not integrated into z196 power. Power for example, comes off wall power.

The key design criteria for zBX are:

- CPC availability and integrity
- Independent life cycles
  - Asynchronously upgradable
  - Enhancements not tied to CPC HW
- Data availability not tied to a single CPC. Shareable between CPCs
- Growth
- System z service
4.2 BladeCenter chassis

In Figure 4-3, you see a BladeCenter chassis with its major components. There are up to two BladeCenter Chassis in each rack with:

- Up to 14 blades1 (POWER7 or x86). There are 14 blade server slots (BS01 to BS14) available. Each slot is capable of housing any zBX supported blades. In the future, certain supported blade types will be double-wide. Slot 14 cannot hold a double-wide blade. Blades must be sequentially plugged into each BladeCenter (first slot and go linearly out; slots must not be skipped).
- Advance management modules (AMM)
- Ethernet switch modules (ESM)
- High speed switch (HSS) modules
- 8 Gbps (800 MB/sec) Fibre Channel switches for connectivity of client supplied disks
- Blower modules
4.3 Blades by function

The blades in a zBX can be divided into two groups:

- Select IBM blades, to run the customer current business as normal applications. Here we have two types:
  - AIX System p® (POWER7) blades, that may physically migrate to zBX or may be acquired by the customer through existing channels or through IBM
  - Linux on IBM x86 (SOD)
- Optimizer blades, to download specific functions from z196 to zBX
4.4 The blade types

☐ Power-7

☐ Smart Analytics Optimizer

☐ IBM x86 Blades (SOD)

☐ WebSphere DataPower Appliance

*Figure 4-5 The blades*

**The blades**

In a zBX there are three optional types of blades:

- POWER7
- WebSphere DataPower® Appliance (statement of direction - SOD)
- X86 (statement of direction - SOD)

They are covered in detail in the next foils.
4.5 Blades data warehouse roles

The blades can play two different roles:

- **Coordinator node**
  Coordinates with which worker nodes will work in parallel on a specific task or workload.

- **Worker node**
  Blades work together to solve a large problem.
4.6 POWER7 blades

These are normal blades used by customers as a general-purpose computing platform:

- Up to 112 blades
  - 14 blades per BladeCenter
  - 2 BladeCenters per rack
  - 4 racks per zBX Model 2
- Managed by the IBM zEnterprise Unified Resource Manager
- Take advantage of the power of the IBM industry-leading UNIX operating system, AIX
- Virtualized with firmware-supplied hypervisor (LIC), the PowerVM® Enterprise Edition
- Entitled through System z firmware
- Performance and energy efficiency
  - Single-wide 8-core.
  - POWER7 (3.0GHz) processor-based blades automatically optimize performance.
  - Ideal for highly virtualized environments with demanding commercial workload performance.
  - 300 GB HDD internal disk.
  - Virtualization performance and scalability.
4.7 WebSphere datapower appliance blades

WebSphere datapower appliance blades
The IBM WebSphere datapower appliance (SOD) integrated in the zEnterprise System is a quite different implementation that can help simplify, govern, and enhance the security of XML parsing and IT services by providing connectivity, gateway functions, data transformation, protocol bridging, and intelligent load distribution. More details:

- Deep-content routing and data aggregation
- SOA stack acceleration
- Application-layer security and threat protection
  - VLAN support provides enforced isolation of network traffic with secure private networks and integration with RACF security.
- Protocol and message bridging
- Blade Hardware Management
  - Monitoring of hardware with “call home” for current or expected problems and support by System z Service Support Representative.
- Energy Monitoring and Management of DataPower Appliance
- DataPower Firmware Load and Update
- HMC Console Integration
- Dynamic Load Balancing
- Virtual network provisioning
- Monitoring and reporting
  Monitoring rolled into System z environment from single console. Time synchronization with System z. Consistent change management with Unified Resource Manager.
- DataPower failure recovery and restart
4.8 Nodes and ensembles

Together, a z196, plus zBX (with up to 112 blades), and Unified Resource Manager constitute a node in a zEnterprise ensemble.

A zEnterprise ensemble can have a maximum of eight nodes, comprised of up to eight z196 CPCs and 896 blades (housed in eight zBXs). The ensemble has dedicated integrated networks for management and data and the Unified Resource Manager functions. Then, a zEnterprise ensemble is a collection of highly virtualized heterogeneous systems, managed as a single logical entity, where diverse workloads can be deployed.

Figure 4-9 shows an ensemble with two z/196s, each with a zBX(2458-002).
4.9 zBX networking and connectivity

*zBX network and connectivity*

zBX and z196 are connected through two LANs, respectively the INMN and IEDN. These networks are exclusive and fully isolated for data traffic between the zBX and the z196.

**Intra-ensemble data network (IEDN)**

The IEDN provides private, secure VLAN-capable network 10 GbE high speed data paths between all elements of a zEnterprise ensemble.

Allows virtual servers on each blade to share data with the other applications throughout the ensemble. The IEDN connects the zBX to a maximum of eight z196 servers.

Each z196 (2817) CPC must have two OSA-Express3 10GbE ports (for redundancy) connected to the zbx (2458). The 2458 can have a maximum of 16 physical IEDN connections (that is, eight pairs of OSA-Express3 10 GbE ports).

They should be connected to two switches in TOR of rack B.

For availability, the two ports should be spread across two different OSA-Express3 10 GbE cards/features within the same CPC, but in different I/O domains if possible. The 2458 should be installed adjacent to the controlling z196 CPC for service requirements.
Intranode management network (INMN)

Communication across the INMN is exclusively for the purpose of enabling the UnifiedResource Manager of the HMC to perform its various management disciplines (for example, performance management, network virtualization management, or energy management) for the node. Then, INMN connects the HMC of the z196 to the hypervisor, optimizer, and guest management agents within the node in the blades. However, virtual servers cannot access the INMN.

Each z196 (2817) CPC must have two OSA-Express3 1000BASE-T ports (for redundancy) connected to the zbX (2458).

The INMN provides connectivity to the Management Module in the BladeCenter Chassis and the BladeCenter Chassis Ethernet switch network. INMNs in different nodes are not connected to each other. They should be connected to two switches in TOR of rack B.
4.10 Hardware management consoles (HMC)

The zEnterprise System Hardware Management Console (HMC) that has management responsibility for a particular zEnterprise ensemble is called a primary HMC. Only one primary HMC is active for a given ensemble. This HMC has an alternate HMC to provide redundancy; it is not available for use until it becomes the primary HMC in a failover situation.

To manage ensemble resources, the primary HMC for that ensemble must be used. A primary HMC can of course perform all other HMC functions.

An HMC network configuration consists of the following elements:

- Primary and alternate HMCs must be connected to the same LAN segment.
  - The z196 can be on a different LAN segment.
- One pair of HMCs (primary and alternate) is required per ensemble.
  - The primary HMC can control up to eight z196 with one zBX each.
- The primary and alternate HMCs use Category 6 Ethernet cables.
  - The maximum distance is 100 meters, but it can be extended with an Ethernet switch.
- An additional 16-port Ethernet switch (FC 0070) can be ordered for the HMC network.
- A zEnterprise ensemble can be configured without zBX (using only z196 nodes).
z/Enterprise Unified Resource Manager

The zEnterprise System consists of three components that work together to bring you the revolutionary best in class systems and software technologies. It is a system of systems that unifies IT for predictable service delivery.

▶ The IBM zEnterprise 196 (z196) is a fast and scalable system.

Ideal for what you run today: large scale data and transaction serving and mission critical applications. Furthermore, following the 10-year anniversary of Linux on the mainframe (2010), we understand it to be the most efficient platform for large-scale Linux consolidation.

▶ The zEnterprise BladeCenter Extension (zBX).

The zBX is a new System z infrastructure built to house our high performance optimizer and select IBM POWER7 and System x86 general purpose blades. Blades allow us to expand our application portfolio with more AIX and Linux applications. Connectivity between the z196 and the zBX is with dedicated high performance private networks, one for data and one for support.

▶ IBM zEnterprise Unified Resource Manager (zManager).

For many customers it is not a new scenario to have power blades connect through the network with mainframes. The big impact of the IBM announcement of zEnterprise is the zManager. The zManager is the revolutionary piece of this solution. It is really the glue that brings this all together and represents the real innovation (in addition to the tremendous innovation in the z196). It is part of the IBM System Director family, and provides platform, hardware and workload management. zManager unifies the management of resources and allows the extending of System z governance across the infrastructure.

The use of Linux on System z is growing rapidly and the zManager makes deploying a workload on Linux on System z much easier than before.
5.1 Unified resource manager introduction

Figure 5-1  Unified resource manager major functions

- Defining and managing virtual environments. This includes:
  - I/O configuration
  - LPs (for PR/SM)
  - Virtual machines (for z/VM)
  - Virtual servers (for POWER/VM)
  - Virtualized LANs

- Defining and managing workloads and policies

- Receiving/applying corrections and upgrades to the LIC

- Performing temporary and definitive z196 capacity upgrades

Unified resource manager (zManager) introduction

zManager is an integrated System z management software facility for zEnterprise platform management. zManager unifies the management of resources across these different architectures through the exploitation of virtualization technology—initially LPAR, PowerVM and z/VM—to enable the creation and management of virtual servers.

Through the zManager, the zEnterprise system is managed as an ensemble, a single pool of resources, integrating system and workload management across the multisystem, multitier, multiarchitecture environment.

zManager code is located in the Support Element (SE) and in the Hardware Management Console (HMC); its major functions are grouped as follows:

- Defining and managing virtual environments
- This includes the automatic discovery as well as the definition of I/O and other hardware components across z196 and zBX, and the definition and management of LPs (for PR/SM), virtual machines (for z/VM), virtual servers (for PowerVM), and virtualized LANs.
- Defining and managing workloads and workload policies
- Receiving and applying corrections and upgrades to the Licensed Internal Code
- Performing temporary and definitive z196 capacity upgrades.

These functions in support of an ensemble are provided by the Hardware Management Console (HMC) and Support Elements (SE) and exploit the intra-ensemble data network (IEDN). Figure 5-2 on page 319 shows the specific functions executed by zManager.
5.2 Refreshing the ensemble concept

Nodes and ensemble concept
Together, z196, and optionally zBX (with up to 112 blades), and Unified Resource Manager constitute a node in a zEnterprise ensemble as a single logical entity, where diverse workloads can be deployed. A node is also called a member in zManager jargon. Refer to “HMC zManager panel” on page 320.

A zEnterprise ensemble is a set of nodes. It can have a maximum of eight nodes, comprised of up to eight z196 servers and 896 blades (housed in eight zBXs). The ensemble has dedicated integrated networks for management and data and the Unified Resource Manager functions. So a zEnterprise ensemble is a collection of highly virtualized heterogeneous systems, managed by the HMC zManager panel.

In Figure 5-2 IEDN stands for intra-ensemble data network and INMN stands for intra-node management network (INMN). These are private networks connecting nodes with nodes (in one ensemble) and connecting node components.
In this panel you can see the icons associated with the ensemble entities. From here you can manage the ensemble entities, such as:

- Ensemble name
- Hypervisors such as LPAR, z/VM and PowerVM
- Members - a member is a z196 with or without a zBX, or in other words a node
- Virtual Servers is a generic name for operating systems running under the hypervisors
- Workloads - a modification of a z/OS WLM concept used by Platform Performance Manager (PPM) policy. PPM is a component of zManager.
- zBX BladeCenter
- zBX blades
5.3 zManager location in zEnterprise

zManager code is located in HMC and in SE. The lines and connected boxes in Figure 5-4 represent the zManager topology in a zEnterprise system. Solid lines (from the SE to other parts) connect zManager with its agents and the private support network (intra-node network management). The dotted line shows that z/OS and zLinux offer monitor capabilities through agents provided to zManager. The thick double arrow line on the bottom is the private data network (IEDN). Refer to “PPM agents” on page 358 to learn more about zManager agents.

In this chapter we explain the following topics, all related to zManager:

- How the hypervisors are treated by zManager. The hypervisors are PS/SM, z/VM, PowerVM and there will be a hypervisor for System x (SOD).
- Support Element (SE) has been extended to orchestrate hardware operational control for the blades, as well as for the entire hardware topology (configuration, problem management, call home, and others).
- Blades are optional, and you get the zManager capabilities with system z196 (do not need a zBX to get initial value on zEnterprise).
- The role of the HMC is being extended, and is evolving to be a full function platform management appliance. The HMC represents the "management access point" for all zManager management functions.
- zManager has "workload awareness" where workloads consist of virtual servers.
- Virtual server processor management.

Figure 5-4  zManager location in zEnterprise
5.4 zManager major roles

As pictured in Figure 5-5, the major zManager roles are managing the following items:

- Hypervisors
- Energy
- Operations
- Networks
- Virtual servers
- Performance

In the next foils, those items are explained in details.
5.5 zManager hypervisors and energy

Unified resource manager details
Here, we offer more details about the zManager abilities and how they can be divided.

Hypervisor management
Blades are virtualized, which means that their Power7 RISC processors run hypervisors (not customer-accessible) and support virtual servers for application deployment. Hypervisor management is a new HMC discipline that manages the blade hypervisors as firmware (LIC code), loading them automatically and encapsulating their management interfaces. Through these interfaces, hypervisors can be deployed, initialized, started, stopped, and listed; updated and repaired; monitored (through the System Activity Display frame); used to hold customer ISO images; used to create virtual networks for interconnecting virtual servers; and configured to allow agents in virtual server operating systems to communicate with manager functions running in the hypervisor or in proxies.

Some of these functions also apply to z hypervisors, such as LPAR and z/VM.

In general terms, this zManager function manages and controls communication between virtual server operating systems and the hypervisor through:

- Integrated deployment, configuration and initialization of hypervisors. Hypervisors (except z/VM) are shipped and serviced as LIC firmware.
- Creation of virtual networks.
- Basic hypervisor management tasks, such as start, stop, and query and list.
- Update and repair hypervisors running in a blade.
- Monitor hypervisors and their resource use through the System Activity Display (SAD) frame at HMC. The resources might be processor, I/O channels, and memory consumption.
- Allow agents (deployed code) running in virtual server operating systems to communicate with a manager running in the hypervisor or the hypervisor management stack. More on agents in “PPM components” on page 354.
- Management of ISO images.

**Energy management**

The energy management strategy for the server is not to go after fine-grained power savings by dynamically and automatically varying the performance of the system. Rather, the objective here is to deliver energy efficient features, energy management functions, and very advanced management solutions that clearly demonstrate value for System z.

Then, zManager is able to execute the following functions related to energy:

- Static power savings and capping.
- Monitoring and trend reporting of processor energy efficiency, through the power estimation tool.
- Ability to query maximum potential power.

Refer to “More details about energy management” on page 326.
5.6 Energy SAD frame

In this frame you might monitor the actual power consumption in volts and the current temperature in Celsius degrees through the system activity display (SAD) frame, a panel of HMC, as shown in Figure 5-7.
5.7 More details about energy management

Energy management
In this section we discuss the elements of energy management in areas of tooling to help you understand the requirement for power and cooling, monitoring and trending, and reducing power consumption. The energy management structure for the server is shown in Figure 5-8.

There is an LIC in the SE (Energy Management) that monitors the temperature and electrical energy metrics managed by the hardware component Power Sub-system Controller. A GUI is defined in such code that allows the exhibition through zManager of such figures on the HMC/SE Energy Monitoring Dashboard panel.

The data captured in the SE is sent to another LIC located in the HMC with the same name (Energy Management). This second LIC offers, through an externalized API, the same data (temperature and electrical energy). The contents made available in such API are described in “Energy data available from HMC” on page 327.

There is an optional software product named IBM Systems Director Active Energy Manager™, that collects such information. Refer to “Systems director active energy manager” on page 328.
5.8 Energy data available from HMC

The following data is available from System z HMC panels:

- System name, machine type, model, serial number, firmware level
- Ambient temperature
- Exhaust temperature
- Average power usage over a 1-minute period
- Peak power usage over a 1-minute period
- Limited status and configuration information

Energy data available from HMC

The following data is available from System z HMC panels:

- System name, machine type, model, serial number, firmware level
- Ambient temperature
- Exhaust temperature
- Average power usage over a 1-minute period
- Peak power usage over a 1-minute period
- Limited status and configuration information. This information helps to explain changes to the power consumption, called Events, which can be:
  - Changes in fan speed
  - MRU failures
  - Changes between power-off, power-on, and IML-complete states
  - Number of books and I/O cages
  - CBU records expiration(s)
5.9 Systems director active energy manager

IBM systems director active energy manager

It is an energy management solution building block that returns true control of energy costs to the customer. Active Energy Manager is an industry-leading cornerstone of the IBM energy management framework and is part of the IBM Cool Blue® portfolio.

Active Energy Manager Version 4.1.1 is a plug-in to IBM Systems Director Version 6.1 and is available for installation on Linux on System z. It can also run on Windows, Linux on IBM System x, and Linux on IBM Power Systems™.

So Active Energy Manager is a management software tool that can provide a single view of the actual power usage across multiple platforms as opposed to the benchmarked or rated power consumption. It can effectively monitor and control power in the data center at the system, chassis, or rack level. By enabling these power management technologies, data center managers can more effectively power-manage their systems while lowering the cost of computing.

Figure 5-10 shows a dashboard panel example produced by the Active Energy Manager.
5.10 zManager operations control

Operations control
Operational controls are management functions executed by zManager that deal with the fundamental hardware components of the system. It is constituted of the following disciplines:

- Change management
- Problem management
- Guided repair and verification
- Configuration management
- Operation management
- Performance monitoring
- Business management
- Ensemble management

In the next foils, these disciplines are detailed.
5.11 Change management functions

- View firmware information (BladeCenters and blades)
- Retrieve firmware changes
- Change firmware levels
- Backup/restore critical data

*Figure 5-12  Change management functions*

**Change management**

Change Management has been extended from z to deal with blade and BladeCenter firmware, enabling current levels to be displayed, new levels to be retrieved from IBM and applied under customer control. As well, critical configuration data is backed up and restored as necessary to ensure the continuity of the configuration across blade replacements. Change Management has the following activities:

- View firmware information (for BladeCenters and blades)
- Retrieve firmware changes
- Change firmware levels
- Backup/restore critical data. Accelerator blades configuration data backed up as part of System z Support Element backup; restored on replacement of the accelerator.
5.12 Problem management

- Automatic error logging and data collection (FFDC)
- Problem analysis and call home reporting
- View hardware messages
- View open problems
- Manual problem reporting and data collection

Problem management
Problem management is extended from z to the blades to enable automatic logging of error and first-failure data capture information. Blade-related problems are analyzed to determine their severity and are reported to IBM through the call-home mechanism for potential service action. Problems that are system-detected can be managed through these mechanisms, as can customer-initiated problems.

When a service action is required, repair activities are guided by the HMC and the effectiveness of the repair is verified automatically.

Problem management has the following functions:
- Automatic error logging and data collection (FFDC)
- Problem analysis and call home reporting
- Cross platform hardware problem detection, reporting and call home
- View hardware messages
- View open problems
- Manual problem reporting and data collection
Guided repair and verification
The problem determination and repair task is executed as a guided tour through hardware and software elements of the zEnterprise.
5.13 Configuration management

- Physical hardware configuration, backup and restore
- Viewing vital product data
- Edit frame layout
- Frames, switches, BladeCenters
- MES support (add/remove)
- Management enablement
- Capacity on Demand
- Permanent customer-initiated upgrades
- Auto-discovery and configuration support for new resources.

**Figure 5-14 Configuration management**

**Configuration management**

Configuration management has been extended from z to handle vital product data for optimizers and blades, determining what zBX resources are entitled to be powered on and managed. The layout of the zBX frame can be displayed and managed (that is, for MES handling), including frames, switches, and BladeCenters. Capacity on Demand support enables permanent customer-initiated upgrades to be applied. It has the following functions:

- Physical hardware configuration, backup and restore
- Viewing vital product data
- Edit frame layout
- Frames, switches, BladeCenters
- MES support (add/remove)
- Management enablement
- Capacity on Demand
- Permanent customer-initiated upgrades
- Auto-discovery and configuration support for new resources.
5.14 zManager HMC configuration panel

HMC configuration panel
Figure 5-15 shows the zManager Wizard configuration panel for a PowerVM virtual server.

It is used for assigning hypervisor virtual processors to the virtual server. Wizard is an old z HMC tool to help customers define POR configurations, LPAR profiles through HMC.

The Wizard interface is exploited by zManager.

In Figure 5-15, one virtual processor is being assigned to a virtual server.
5.15 Operations management

- Blade Power On and Power Off
- Upstream SNMP/CIM API Automation Management
- Event notification (based on logged events or state changes)
- Scheduled operations
- Time synchronization
- Operational network settings (IP address, group name, role)
- Launch Full Device Console

Operations management
Operations management allows blade power to be controlled and extends the existing HMC programming interfaces to include operations on zBX resources. Blade-related events can be used as a basis for email and other notifications to support personnel. Functions such as firmware updates and blade activation can be managed through scheduled operations, and a time server enables blade resources to synchronize with a central time source. Settings for the operational network can also be managed, and a console can be launched to provide direct access to blades. It has the following functions:

- Blade Power On and Power Off
- Upstream SNMP/CIM API Automation Management
- Event notification (based on logged events or state changes)
- Scheduled operations, such as Firmware Update, Activate, Deactivate
- Time synchronization
- Operational network settings (IP address, group name, role)
- Launch Full Device Console.
5.16 Performance monitoring and business management

- **Performance Monitoring**
  - System Activity Display - SAD frame
  - Energy Consumption and Temperature Monitoring
  - Transmit System Availability Data (TSAD)
  - Delivery of system activity using new interface

- **Business management**
  - User Management and automatic service network configuration
  - Auditing and documentation
  - Device Status and Details

---

**Performance monitoring**

The System Activity Display (SAD) frame is extended to show the performance of zBX resources, including processor, memory, I/O, and network activity. As well, energy consumption and temperature are reported for these components. Their availability information is included in the data provided by the Transmit System Availability Data function. Do not mix up this function of zManager inserted in Operations Control with another role also from zManager called Platform Performance Manager (PPM) described in “zEnterprise platform performance manager” on page 341. Performance Monitoring has the following functions:

- System Activity Display - SAD frame for processor, memory, I/O, and network
- Energy Consumption and Temperature Monitoring
- Transmit System Availability Data (TSAD)
- Delivery of system activity using a new interface

**Business management**

Business management functions including user management, auditing, device status, service network configuration, and documentation, are updated to incorporate zBX components. It has the following items:

- User management and automatic service network configuration
- Auditing and documentation
- Device status and details
5.17 Ensemble management

Ensemble management
In order to manage zBX resources or activate enhanced management functions for Linux on System z on z/VM, the zEnterprise System must be configured as an “ensemble”. This involves designating primary and alternate ensemble management HMC and selecting the nodes of the ensemble (members) that belong to the ensemble. As we will see, this enables high-speed private and secure network connections between ensemble member components, as well as providing a broader scope for the management of the member systems.

Figure 5-18 shows a zManager HMC ensemble panel used for adding new members (PXBONZAI) to an already existent ensemble (AGPARTS).
5.18 zManager performance, virtual life cycle and networks

Last three zManager roles
Here, we cover the last three roles of zManager, as pictured in the pie of Figure 5-19.

Performance management
Performance management is also called Platform Performance Manager (PPM). It is in charge of goal-oriented resource monitoring, management, and reporting across the zEnterprise ensemble. It very much has the concepts of the former EWLM. It is covered with more details in “zEnterprise platform performance manager” on page 341. Platform Performance Manager has the following functions:

- Workload awareness
- Wizard-driven management of resources in accordance with specified business service level objectives. Wizard is a user-friendly HMC functionality.
- HMC provides a single consolidated and consistent view of resources.
- Monitor resource use within the context of a business workload.
- Define workloads and associated performance policies.

Virtual server life cycle management
Virtual server life cycle management, enabling directed and dynamic virtual server provisioning across all hypervisors from a single, uniform point of control. This includes
integrated storage and network configuration, and ensemble membership integrity. It has the following functions:

- Single view of virtualization across platforms
- Ability to deploy multiple, cross-platform virtual servers within minutes
- Management of virtual networks including access control
5.19 Network management

Network management manages virtual networks including access control, such as adding or removing virtual networks.

It provides management of Guest LANs, Virtual Switches, and VLAN.

In Figure 5-20, we have an example through zManager HMC panels of adding a VLAN to an ensemble, which has a VLAN ID of 11.
5.20 zEnterprise platform performance manager

Platform performance manager (PPM) is the zManager component responsible for performance goal-oriented resource monitor, management, and reporting across the zEnterprise ensemble. It has the following functions:

- Define and implement a goal-oriented management policy, extending this goal-oriented approach of z/OS WLM to platform-managed resources.
- Monitor and report workload-based information about management policy. Provide analysts with data to understand whether goals are being met.
- Monitor and manage the platform resources, with a common approach, across the zEnterprise.
- Orchestrator® autonomic management of resources across virtual servers, providing:
  - Intelligent Resource Director like function across the zEnterprise
  - That management function will evolve over time
- Push management directives to SE, hypervisors, and OS agents as required across the zEnterprise.
- Integrating of HMC console support:
  - Integrated user interface (UI) for monitoring, displaying of workload topology, status alerts
  - Definition of performance management goals and policy administration
Functionality integrated into the zManager
- Code structured and packaged as System z firmware
- Inter-component communication over trusted internal platform management network
5.21 PPM virtual servers

PPM Virtual Servers

- zEnterprise Platform Performance Management associates performance goals to the following types of operating systems and accelerators running in virtual servers:
  - z196 logical partitions (LPs) running z/OS
  - z196 z/VM guests running Linux
  - zBX virtual servers running AIX on the POWER7 blades
  - zBX virtual servers running Windows or Linux on System x on System x blades

PPM virtual servers

In the zEnterprise system, the logical partitions (LP) running in the z196 may be called virtual servers. Normally, the z/OS LPs have the workload performance defined, monitored and adjusted through the z/OS Workload Manager (z/OS WLM), but that is only part of the picture.

Workloads running on z/OS are classified into z/OS WLM service classes and have objectives defined to express how they must perform. These definitions are used by z/OS WLM to manage the work across all z/OS systems of a sysplex environment.

zEnterprise PPM functions provide a more comprehensive performance management by making it possible to extend this objective-oriented concept to the ensemble. Now with PPM, we can set performance objectives and track the performance of any virtual server in the ensemble. Hardware and software resources are also grouped into workloads, and performance objectives are defined for the following virtual servers:

- z196 logical partitions (LPs) running z/OS
- z196 z/VM guests running Linux
- zBX virtual servers running AIX in the POWER7 blades
- zBX virtual servers running Windows or Linux on System x on System x blades

Orchestration of autonomic management of resources across virtual servers is performed by:

- Workload balancing.
Providing Intelligent Resource Director (IRD) like function across the zEnterprise system, where weights can be dynamically changed.

Leveraging of virtual server mobility to achieve performance objectives.

PPM then includes the following capabilities:

- Performance monitoring and reporting functions.
- Virtual server processor management provides the ability to manage processor resources across.
- Virtual servers based on an objective-oriented performance policy.
- Cooperative management enables z/OS WLM to manage the z/OS segments of multitiered workloads. It considers the overall workload objectives.
- Load balancing guidance enables the distribution of incoming work to best achieve the objectives defined in the workload performance policies.
5.22 Virtual server definition

**Defining a virtual server**

Platform Performance Manager (PPM) deals with virtual server (z/OS LPs, z/VM virtual machines and PowerVM virtual servers) performance in order to accomplish the goals defined in the policy.

You now can use the New Virtual Server Wizard, or select a hypervisor from the Hypervisors tab of the ensemble, to create a new virtual server. After definitions have been made, the summary of such a definition looks like Figure 5-23. Wizard is an old z HMC tool to help customers to define Power on Reset (POR) configurations, and LPAR profiles through HMC user friendly panels.
5.23 z/OS WLM main terms

- Service definition
- Policies
- Workloads
- Service class
- Service class periods with goals and importance
- Transactions
- Report class
- Classification rules

Figure 5-24  z/OS WLM main terms

z/OS WLM main terms

PPM is a sort of z/OS WLM running outside of the z/OS boundaries, like the former EWLM. PPM uses quite the same terminology as z/OS WLM, when z/OS WLM is taking care of transactions running in a Parallel Sysplex. It is important here to recap some of the main terms and associated constructs used in a z/OS WLM service definition because the same terms are also used in the PPM policy, at HMC. From now on in this chapter, we refer to z/OS WLM as WLM only:

- Service definition is a set of policies stored in the WLM couple data set.
- Policy is a set of WLM constructs (such as the ones defined below) that associate incoming transactions and address spaces with goals. A service definition might have several policies, but just one is active in a Parallel Sysplex at a time. All the workloads (and the service classes) are shared between all the defined policies. The difference between policies is the numeric value of goals and importance, but in the same service classes.
- Workload: arbitrary name to group various service classes together for reporting and accounting purposes only.
- Service class: a set of service class periods.
- Service class periods: where you define the goal and the importance for a specific type of transactions or address spaces. The goal indicates indirectly the desired level of service that WLM uses to determine the amount of resources to give to a set of transactions. The importance tells WLM how important it is to accomplish such a goal to the business. The periods of a service class are designed to protect trivial transactions.
- Transactions: z/OS business transactions (such as CICS, WebSphere Application Server) or z/OS address spaces associated with a service class.
- Report class: aggregate a subset of transactions from a service class, for reporting purposes only.
- Classification rules: rules to assign incoming transactions and starting address spaces to service classes and consequently to goals.
5.24 Intelligent resource director review

Intelligent resource director (IRD) review
Platform Performance Manager (PPM) is in charge of performance goal-oriented resource monitoring, management, and reporting across zEnterprise ensemble. These goals are applied not to transactions but to virtual servers (or virtual machines) running non-z/OS operating systems such as zLinux and AIX.

This function is similar to a current WLM IRD function named Vary Weight CPU Management that also applies to z/VM Linux logical partitions running in an LPAR cluster. This is the reason why we are reviewing such concepts before tackling PPM.

IRD is a WLM/LPAR function that applies to an IRD LPAR cluster.

IRD LPAR cluster
An IRD LPAR cluster is a set of z/OS in the same CPC and in the same Parallel Sysplex. By a special definition, it may include Linux LP or z/VM Linux logical partitions located in the same CPC as z/OS. Then in this case zLinux may belong to an IRD LPAR cluster.

Figure 5-25 shows two IRD LPAR clusters (one and two). The number two contains two z/OS and one zLinux. To implement WLM Vary Weight CPU Management, WLM needs to allocate a CF structure, as pictured in the figure.

IRD Vary Weight Cpu Management
This function swaps LP weights among LPs in an IRD LPAR cluster.
One donor’s weight is subtracted from LP A, and added to the weight of LP B (receiver). The trigger for such action is that in the receiver LP B there are transactions with affinity to this LP B, running in a service class period with high importance, unhappy, and suffering processor delays.

The installation may limit, by a range, the minimum and maximum value of the weight in each LP in the LPAR cluster.

Sysplex name and system names must be entered on HMC (to add to an LPAR cluster). Using system names allows a non-z/OS LP (zLinux) to be associated with an LPAR cluster.
5.25 RD for a zLinux logical partition

Then, the IRD Vary Weight CPU Management functionality is extended across non-z/OS LPs. WLM IRD can shift weight from either z/OS or zLinux LP to another z/OS or zLinux LP in the same IRD LPAR cluster, to meet goals. Note that the goal applies to the full zLinux logical partition. It does not apply to:

- zLinux internal transactions.
- Specific virtual machines running zLinux under z/VM.

In other words, the zLinux LP is treated as a single entity, similar to a long-running task, with an importance and a type of goal that can be execution velocity only (no response time or discretionary type of goal), in the policy.

The association between the zLinux LP and one service class (with just one period) is made in the classification rules through the subsystem name SYSH. In Figure 5-26, note that the zLinux LP named zVM1 is associated with the service class SCZVM. And in the other WLM policy panel, SCZVM has an Execution Velocity goal of 40% and importance 3.

The basic IRD requirements are: shared CPs, non LPAR capped, and enabled for IRD Vary Weight Management in HMC. As you can see, because the requirement of sharing processor, only a few customers have used this function because the huge majority exploits non-IRD supported IFL processors for zLinux LPs.
5.26 RMF and IRD zLinux implementation

Figure 5-27 shows an RMF monitor III report named SYSSUM.

In each line, we have a service class period data information.

In the last line, the SCZVM service class is associated with a zLinux logical partition named zVM1, as defined in the classification rules. Refer to “RD for a zLinux logical partition” on page 350.

The Execution Velocity goal is 40% and the current value is 20%. So this logical partition is unhappy by WLM IRD terms. Note that the performance index is 2.0, confirming that the goal was not achieved.

How IRD zLinux works
This topic is very important because the way that IRD works is similar to PPM.

The Execution Velocity goal for zLinux Vary Weight CPU Management has the following formula:

\[
\text{CPU}_{\text{Using}} \times 100 / (\text{CPU}_{\text{Using}} + \text{CPU}_{\text{Delay}})
\]

The \text{CPU}_{\text{Using}} figure is obtained by IRD from the LPAR hypervisor and is numerically equal to the amount of processor seconds executed in this LP along a certain time interval.
The CPU_Delay figure is obtained by IRD from the LPAR hypervisor and is numerically equal to the amount of processor seconds that the logical processors are ready in this LP along a certain time interval. Ready here means that they want to have a physical processor but they cannot get one, so they are being delayed.

It is now business as usual, when the service class period SCZVM is the most important and the most unhappy, and WLM tries to help by declaring it as processor receiver. Another LP (it could be a z/OS LP) is declared processor donor. Then, the LPAR hypervisor is invoked in order to increase the weight of the zLinux LP and to decrease the weight of the donor LP.
5.27 PPM wizard welcome panel

In Figure 5-28, we see the PPM Wizard welcome panel, with which we are able to define PPM policies.
5.28 PPM components

Now we are almost ready to better cover PPM.

PPM consists of several components physically spread across different parts of the zEnterprise complex:

- Running in the HMC with the following functions:
  - Use HMC as a management server and as a console
  - Getting ensemble-wide aggregation of performance data
  - User interface for defining workloads, performance policy, and reporting data
  - Pushing management directives to all the nodes of an ensemble

- Running in the Support Element (SE) with the following functions:
  - Getting node (or CPC) level aggregation of performance data
  - Pushing management directives to all the hypervisors in the node

- Running in hypervisors (PR/SM, z/VM and PowerVM) as an agent, with the following functions:
  - Monitoring goal defined in performance policy and performing dynamic resource management to achieve the performance goal where applicable
  - Collecting virtual server statistics from a hypervisor and guest platform management providers. Pushing aggregated metrics to SE.
Running in virtual servers (or LPs or virtual machines) as an agent, with the following functions:

- If Optional Guest Platform Management Provider (GPMP) software is deployed in a virtual server as an agent, then additional reporting data is available.
- Collecting monitoring data from the operating system and Application response measurement (ARM) instrumented applications and pushing to hypervisors.

Refer to “PPM agents” on page 358 for more information about GPMP, and refer to “Application response measurement (ARM)” on page 359 for more information about ARM.
5.29 Differences between PPM and z/OS WLM

- For PPM:
  
  \[ \text{VELOCITY\%} = \frac{\text{USING\_PROC} \times 100}{\text{USING\_PROC} + \text{DELAY\_PROC}} \]

- For WLM:
  
  \[ \text{VELOCITY\%} = \frac{(\text{USING\_PROC} + \text{USING\_IO}) \times 100}{(\text{USING\_PROC} + \text{USING\_IO} + \text{DELAY\_PROC} + \text{DELAY\_IO} + \text{DELAY\_STORAGE} + \text{DELAY\_SERVER ADDRESS SPACES})} \]

Figure 5-30  Velocity formulas for PPM and WLM

Differences between WLM and PPM WLM

We now pinpoint the major differences between WLM (without taking IRD in consideration) and PPM:

- PPM classification rules do not apply to transactions, they apply to virtual servers (operating systems) only. The reason is that in a UNIX environment there is just one type of transaction per virtual server. WLM applies to z/OS transactions and ASs with many qualifiers.

- PPM only controls processors. WLM controls mainly PUs (processor, zIIP and zAAP), I/O operations, central storage use and the amount of server address spaces.

- The PPM service class has velocity and discretionary goals. WLM has velocity, response time and discretionary goals. However, the two discretionary goals do not have the same meaning. In PPM discretionary means no goal, in WLM it means “do the best you can.”

- The formulas for velocity are not the same. Refer to Figure 5-30. Another difference between the formulas is that the values for PPM are real and produced by the hypervisor looking at the processor in the virtual server. The values in WLM formula are WLM sample values of service time and queue time of several resources per transaction or per address space.

- In the PPM policy, at the velocity declaration, you use qualifying words such as high and low. In the WLM policy, you use percentages between 100% and 0%.
In PPM, in order to reach goals, you change the weights (or entitlement capacity) through hypervisors. WLM changes mainly weights (in logical partitions only), z/OS dispatching priorities, I/O priorities, storage protections thresholds, and the number of server address spaces (for certain type of transactions).

Both PPM and WLM have the workload concept but with a different meaning. For PPM, a workload is a set of virtual servers to be associated with one service class, which looks in WLM terms as a classification group. For WLM, a workload is just a set of service classes put together for accounting and report reasons.
5.30 PPM agents

PPM agents are special optional code in the ensemble. They enable a data flow communication between the virtual server operating systems and hypervisors with zManager (PPM in particular). They are connected through the private support network, the intra-node management network (INMN).

Optional Guest Platform Management Provider (GPMP) is an optional lightweight agent deployed in virtual servers only. GPMP provides:

- Additional monitoring data. With instrumented middle ware support (ARM), GPMP provides metrics that allow detailed transaction topology, as transaction hops through heterogeneous platforms in zEnterprise. Refer to “Application response measurement (ARM)” on page 359 for more about ARM.

- Classification of virtual servers using additional attributes such as host name, system name, and OS level.

GPMP is free and can be installed in z/OS, AIX and Linux.
5.31 Application response measurement (ARM)

Application response measurement (ARM) concepts
ARM is a middleware standard application instrumentation from OpenGroup (Standard V4.0).

In other words, it is a standard way of passing performance information among systems. In a zEnterprise PPM environment, ARM can be used by the agents.

The advantages of having such instrumentation within PPM are:
- Capturing real transaction response time
- Multitiered work request flow across environments
- Relationship with server resources being consumed
- Some reasoning lead to even use such instrumentation for z/OS WLM with z/OS subsystems, such as CICS, DB2, and IMS™
5.32 Virtual server processor management (I)

Here we explain in simple terms how PPM works in order to accomplish its goals. Virtual Server CPU Management is a main component of PPM running at all times. It provides the ability to manage processor resources across virtual servers (operating systems) based on a goal-oriented performance policy.

Virtual Server CPU Management has the following functions:
- Manage processor resources across virtual servers to achieve workload goals
- Detect that a virtual server is part of a workload not achieving goals
- Determine that the virtual server performance can be improved with additional processor resources
- Project impact on all affected workloads of moving processor resources from another virtual server (donor) to this virtual server (receiver)

In other words, Virtual Server CPU Management controls the processor assignment of PowerVM and z/VM through service class goals (Velocity with Using and Delay CPU only). It looks like WLM IRD controlling the weight (or the PowerVM entitlement capacity) in operating systems such as zLinux under z/VM and AIX under PowerVM. Refer to “Intelligent resource director review” on page 348.
In Figure 5-33 on page 360, you can see a PowerVM virtual server (unhappy with the assigned velocity goal) receiving an entitlement capacity raise donation from the other PowerVM virtual server (happy with its assigned velocity goal).

The decision whether to change the weights (or the PowerVM entitlement capacity) is made every second.

All the captured performance data is kept in HMC for 36 hours and looks like SMF 99 type of data, that is, very detailed information about PPM decision logic. It is necessary to define agents in order to get performance data.
5.33 Virtual server CPU management (II)

Here is another example about Virtual Server CPU Management (refer to “Virtual server processor management (I)” on page 360.) logic. In Figure 5-34 is a virtual machine running under z/VM that needs help. The following steps are executed:

- Manage resources across z/VM virtual machines
- Detect that a virtual machine that is part of a workload is not achieving goals
- Determine that the virtual machine performance can be improved with additional processor resources
- Project impact on all affected workloads of moving processor resources from another virtual machine (donor) to this virtual machine (receiver)
5.34 PPM major constructs

Figure 5-35 shows the major PPM constructs and its hierarchy:

- Workloads
- Performance Policies
- Service Classes
- Classification Rules

Observe that these constructs do not have exactly the same meaning as the z/OS WLM constructs, even when having the same name. Refer to “Differences between PPM and z/OS WLM” on page 356 to get more information about such differences.

Starting at “PPM workload concepts” on page 364, you can find a detailed explanation about the constructs pictured in Figure 5-35.
5.35 PPM workload concepts

Workload concepts
The PPM concept of workload is a set of zLinux virtual machines (in z/VM in the z196 LPs), virtual servers (in PowerVM in the zBX blades) and z/OS logical partitions (in PR/SM in the z196).

In Figure 5-36, we see an example: the workload Product Sales encompasses 12 operating system images:

- Two z/OS running in two LPs in one 196
- Six zLinux running in one z/VM LP in the same z196
- Three AIXs running in one PowerVM blade
- One AIX running in another PowerVM blade

A workload can have multiple performance policies associated with it. In Figure 5-36, the Product Sales workload belongs to two policies, Off-Peak and Peak.
5.36 PPM workload definition

Workload definition
In PPM jargon a workload is also called a platform workload.

A platform workload is a grouping mechanism of virtual servers and PowerVM accelerators supporting a cross business application. Each platform workload definition has a given name, and the associated virtual servers and one or more related performance policies. Comparing with WLM in z/OS, it looks like a classification group.

In Figure 5-37, we see a Select Virtual Servers wizard PPM panel:
- The ensemble has defined six virtual servers - five under PowerVM (AIX or accelerator) and one a z/OS under LPAR.
- Three (Buyer1, Buyer2 and z/OS PAR) are selected to form a new defined workload.
- The virtual server Payroll Proc1 is already selected, and belongs to a workload named Payroll.

Workload performance policy defines performance objectives for virtual servers in a workload. It provides a basis for monitoring and management of platform resources used by virtual services in a workload. If different applications have the same performance objectives, it is possible to define only one workload that includes all virtual servers where those applications are going to run. If applications have different performance objectives, more than one workload must be defined. When an ensemble is created, a default workload performance policy applies to all of the virtual servers in that ensemble.
5.37 PPM policy

The PPM policy is conceptually similar to simplified z/OS WLM policy.

In other words, the PPM policy provides the basis for monitoring and management of platform processor resources used by virtual servers.

The installation defines performance goals for ensemble virtual servers contained in a workload, using service classes. These service classes are associated with virtual servers (in workloads) through the classification rules. Just a single policy is active at a given time, but it can be dynamically replaced by another one, through:

- The user interface (UI)
- A time-based schedule. Example: Day shift policy or night shift policy

In Figure 5-38, we see the defined PPM policy called Peak Period.

The workload performance policy structure contains a set of service classes; classification rules mapping each virtual server in the workload to a service class, and the assignment of a service class to a performance objective and an importance value. To provide additional performance objectives besides the default one it is necessary to define one or more workloads for the ensemble.
PPM policy structure
The PPM policy has the following structure:

- Set of virtual servers
- Set of workloads grouping virtual servers
- Classification rules mapping each workload (set of virtual servers) with a service class
- A service class assigning a performance goal (velocity or discretionary) and importance to the associated virtual servers

HMC as a console
Policy creation and editing is performed by the ensemble Hardware Management Console (HMC). There is a wizard that helps in policy creation and editing. HMC provides a repository for policies under development and saved policies. It also contains links to workload-based performance reporting. It has:

- A wizard for policy creation
- A repository for policies under development and saved policies
- Links to workload-based performance reporting
5.38 Service class concepts

- **Assigned through Classification Rules:**
  - Use Virtual Server Name as qualifier to assign Service Class

- **Performance goal:**
  - Velocity: Fastest, Fast, Moderate, Slow, Slowest
  - Discretionary: No performance goal

- **Business Importance:** Highest, High, Medium, Low, Lowest

- **Just one period**

---

**Service class concepts**

The concept of service class in PPM is analogous to that in WLM.

- Service classes are assigned through classification rules:
  - But use only the virtual server name (through workloads) as qualifier to assign service classes

- Service classes have the performance goals. The type of goals cannot be based on response time. The possible types are:
  - Velocity has the same structure as in WLM, but it is not numeric and it is only based in CPU Using and CPU Delay. Refer to “Differences between PPM and z/OS WLM” on page 356. The values are Fastest, Fast, Moderate, Slow, and Slowest.
  - Discretionary - Indicates that there is not a performance goal at all.

- In the case of several unhappy service classes, the importance plays the tie break concept as in WLM; however, the values are not numeric: Highest, High, Medium, Low, Lowest.

- There is just one period per service class.
5.39  Service class definition

In Figure 5-40, you can see the panel used to define a PM service class.

In the example, there are two of them:
- Buyers with velocity Fastest and Importance Highest
- Default with velocity Moderate and Importance Medium

Next, these service classes must be associated with workloads through classification rules.

Refer to “Service class definition” on page 369.
5.40 Classification rules

PPM classification rules
Classification rules are a PPM construct, where you assign workloads made of virtual servers with a service class. As we know, inside the service class we already defined the goal and the importance that such a goal be accomplished.

In Figure 5-41, we see the window where we select the virtual server names (buyer1) to be assigned to an existent service class.
5.41 z/OS WLM agent

The GPMP agent is the interface between the INMN network and z/OS WLM. So there is a communication between PPM and the z/OS WLM agent through this network.

Figure 5-42 shows the GPMP option at option 11 of the z/OS WLM policy.
5.42 Connecting PPM SC with a WLM service class

Connecting PPM SC with WLM SC

PPM provides policy information to WLM about the platform-wide performance goals of the MPP workloads in which the z/OS system is participating.

This information is needed because some of the transactions (CICS, for example) running in a z/OS may participate in a PPM workload. As a consequence, we might have:

- z/OS providing differentiated service to PPM classified work
- Transactions coming to z/OS need to be ARM instrumented
- The WLM service definition needs to map PPM service classes into z/OS WLM service classes. This mapping is done through the WLM policy panel pictured in Figure 5-43, for the subsystem EWLM. In the figure PPM SC Gold_Tra is mapped at WLM SC TC_Gold.
- The PPM service class associated with a transaction is used by WLM to classify a work unit to a different WLM service class. Then WLM manages the resources based on the goal assigned to this specific service class. For example:
  - PPMSCGold is mapped to WLMSCGold with 0.5 second Average Response Time Goal and Importance 1
  - PPMSCSilver class is mapped to WLMSCSilver with 0.5 second Average Response Time Goal and Importance 2.
5.43 z/VM agent role in PPM

Supports the vision that ensemble resources are owned, managed and under access control of the ensemble. This is done through:

- Enhancements to provide function required to permit the zManager to perform z/VM system management tasks and virtual server management tasks in an ensemble node through SMAPIs.
- z/VM support for the connectivity aspects of IEDN and INMN internal networks, that is, to support VLAN, virtual nivs, vswitchn...
- Provide access control for the management of these resources and access control points of guest exploitation of the internal networks.
5.44 PowerVM agent role in PPM

- This management is directed by the HMC and its user interface through the SE
- The firmware is packaged and managed as firmware on the Support Element
- The hypervisor has two bonded network adapters that it uses to provide virtual server access to the intra ensemble data network (IEDN) and the customer network
  - Two other bonded network adapters provide intranode management network (INMN) access

**PowerVM agent role in PPM**

This management is directed by the HMC and its user interface through the SE. Those controls allow an IBM Power Blade to be treated as a closed system and are part of the HMC user interface. In addition to blade hardware and firmware management, the HMC provides support for the creation and management of virtual servers.

The firmware is packaged and managed as firmware on the Support Element, which means that it is loaded automatically at power on, serviced through the System z microcode load process, and managed with a full set of operational controls.

The hypervisor has two bonded network adapters that it uses to provide virtual server access to the intra ensemble data network (IEDN) and the customer network. Two other bonded network adapters provide intranode management network (INMN) access. The hypervisor creates virtual network adapters to enable virtual servers to communicate over these networks. All the networks are enabled for IPv6.
5.45 PPM performance data reporting

One major function of the agents spread across hypervisors and virtual servers is the possibility of having a centralized ensemble approach to exhibit performance data.

In Figure 5-46, we see graphically the variation of the average processor utilization in the virtual server Buyer 1 along a specific time interval.

**Figure 5-46 Performance graphics in HMC**
Chapter 6. System z connectivity

This chapter discusses the connectivity options available to connect the System z servers with I/O control units, other System z servers and the network (LAN and WAN). It also highlights the hardware and software components involved in such connectivity.

Officially the System z family of servers encompasses the z9 EC model, z9 BC model, and the zSeries models.

Input/Output (I/O) channels are components of the Channel Subsystems (CSS). They provide a pipeline through which data is exchanged between servers, or between a server’s external devices usually managed by I/O control units.

The most common attachment to a channel is a control unit (CU) accessed via an Enterprise Systems Connection (ESCON) or Fibre Connection (FICON) channel. The control unit controls I/O devices such as disk and tape drives.

Server-to-server communications are most commonly implemented using InterSystem Channels (ISC), Integrated Cluster Bus (ICB) channels, and channel-to-channel (CTC/FCTC) connections. Internal Coupling (IC) channel, as well as HiperSockets and channel-to-channel connections can be used for communications within a server (between logical partitions).

The Open System Adapter (OSA) provides direct, industry standard LAN and ATM network connectivity and communications in a multivendor networking infrastructure.

This chapter contains the following:
- Connectivity highlights
- Channel subsystem concepts
- Channel type descriptions
6.1 Connectivity overview

I/O channels

A channel (333 MHz in the z9 EC) is in charge of a dialog with the I/O controller. The reason for this dialog is the execution of an I/O operation. A channel is composed of circuitries and buffers located in a card plugged in the z9 EC I/O Cage, plus one adapter.

Channels can only use one type of protocol to “speak” to the controller. The major channel protocols in the z9 EC are: Enterprise Systems Connection (ESCON), Fibre Connection (FICON) Express 2 and Fibre Channel Protocol (FCP).

Adapters can only be connected to links using Fiber Optic Technology. The FCP channel is not supported by z/OS and is used by the Linux operating system running in a z9 EC model server.

Channel connections

Channels are components of the System z Channel Subsystems (CSS). In a sense, they provide a pipeline through which data is exchanged between servers, or between a server and external devices including a network.

Channels connect in the following ways:

- Server-to-external I/O control unit (CU) such as disk and tape drives, accessed via an ESCON or FICON Express or FCP channel.
Server-to-integrated built-in control units, such as:

- Open System Adapter (OSA) Express2 provides direct, industry standard LAN and WAN network connectivity/communications in a multivendor networking infrastructure
- Crypto Express2 provides outboard cryptography; refer to “Cryptography in z196” on page 277 for more information.

Server-to-server, such as:

- z/OS-to-Coupling Facility (or Coupling Facility-to-Coupling Facility):
  - Inter System (ISC) channels - external
  - Integrated Cluster Bus (ICB) channels - external up to 7 meters
  - Internal Coupling (IC) channels - internal
- Operating system-to-operating system:
  - Among z/OSs: Channel-to-channel (CTC/FCTC) channels - external
  - Among Linux and z/OSs and z/VMs: HiperSockets - internal
- Sysplex Timer (ETR)-to-server. Refer to *ABCs of z/OS System Programming Volume 5* for more information about ETR links.
6.2 Multiple Image Facility channels

I/O sharing

I/O sharing was already possible in a pre-EMIF ESCON environment, where multiple MVSs could share control units, devices, and common links through ESCON Directors or even by local attachments in the controllers. Channel assignment, however, was more static in relation to logical partitions (LPs). Channels could only be defined as reconfigurable, enabling them to be removed manually from one LP and attached to another. They were dedicated to one LP at a particular instant (no concurrency), and could not be shared by other LPs.

Multiple Image Facility

The Multiple Image Facility (MIF) enables channel resources to be shared across LPs, within a single channel subsystem (CSS), or across multiple CSSs (if spanned channels). Channels receive the order for executing I/O operations from the z/OSs running in LPs (through an SSCH instruction), flip-flopping from one z/OS to the other. A dedicated channel only serves one LP. A shared channel may serve several LPs, where one I/O is for LP1 and the next I/O may be for LP2. The following channel types can be used for MIF IDs:

- ESCON
- FICON
- OSA-Express
- Coupling Facility links
z9 EC and z990 CSS and MIF channels

The structure of multiple CSSs is an extension to the z/Architecture servers. It provides channel connectivity (up to 256 channels per LP) to the LPs in a manner that is transparent to programs. The multiple CSS structure introduces new components and terminology that differs from previous server generations.

Note the following HCD considerations:

- No LPs can be added until at least one CSS has been defined.
- LPs are now defined to a CSS, not to a server.
- An LP is associated with one CSS only.
- CHPID numbers are unique within an CSS; however, the same CHPID number can be reused within all CSSs.

Sharing a channel across LPs in multiple CSSs is known as “spanning;” it was introduced with the z990 server and is supported in the z9 EC server.

Dynamic addition or deletion of a logical partition name

On the z10 EC, all undefined logical partitions are reserved partitions. They are automatically predefined in the HSA with a name placeholder and a MIF ID. This means that any logical partition can be added or deleted dynamically, without performing a power-on Reset (POR).
6.3 Channel subsystem connectivity

The concept of channel subsystem (CSS) was introduced in z990 servers, and it is available in the z9 EC. The z9 EC supports up to four channel subsystems (and 60 LPs), hence the term “multiple channel subsystem.” The design of the z9 EC offers a considerable increase in processing power, memory size, and I/O connectivity. In support of the larger I/O capability, the channel subsystem has been scaled up correspondingly and the CSS concept facilitates the architectural change that provides more logical partitions and channels than before.

Table 6-1 lists the number of LPs and CHPIDs supported.

Table 6-1  Logical partitions and CHPID numbers support

<table>
<thead>
<tr>
<th>Date</th>
<th>CSS supported</th>
<th>Number of active LPs</th>
<th>Number of defined LPs</th>
<th>Number of server CHPIDs supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>z990 - 06/ 2003</td>
<td>CSS 0, 1</td>
<td>15</td>
<td>30</td>
<td>512</td>
</tr>
<tr>
<td>z990 - 10 2003</td>
<td>CSS 0, 1</td>
<td>30</td>
<td>30</td>
<td>512</td>
</tr>
<tr>
<td>z9 EC</td>
<td>CSS 0, 1, 2, 3</td>
<td>60</td>
<td>60</td>
<td>1024</td>
</tr>
</tbody>
</table>

Figure 6-3 illustrates a number of important points for I/O connectivity:

- Two CSSs are shown; however, the z9 EC allows four CSSs.
An LP is associated with a specific CSS. Also note that LPs have unique names across the complete system. LPAR naming and numbering has become somewhat more complex, as discussed in “LP ID, MIF ID, and spanning concepts” on page 251.

Multiple LPs (up to 15) may be associated with an CSS.

A CHPID is associated with a specific CSS. CHPID numbers are unique within that CSS, but may be reused in other CSSs. (For example, there is a CHPID 80 in both CSSs.) Note that CHPID numbers are arbitrarily selected. For example, we could change CHPID 80 (in either or both CSSs in the illustration) to C3 simply by changing a value in the IOCDS.

A CHPID is associated with a PCHID, and PCHID numbers are unique across the server.

Different channels on a single I/O adapter can be used by different LPs. As shown in Figure 6-3 on page 382, PCHID 0140 is the first channel on the adapter in I/O cage 1, slot 6. PCHID 0141 is the second channel on the same adapter.

Figure 6-3 on page 382 also illustrates the relationship between LPs, CSSs, CHPIDs, and PCHIDs. The figure includes the I/O devices and switches used in an I/OCP example; refer to “I/OCP statements example” on page 258 for more information about this topic.

Multiple subchannel sets (MSSs)

Multiple subchannel sets (MSSs) provide relief for I/O device configurations in large System z10 and System z9 environments. MSS are supported by ESCON, FICON, and z/OS, and allow for increased device connectivity for Parallel Access Volumes (PAVs).

Because the IBM System Storage DS8000 and DS6000 series and the System z10 and System z9 support PAVs, there is symmetry between the server and the storage subsystem. A pragmatic implementation approach can be used for the adoption of MSS in a System z environment.

MSS functionality should not be confused with multiple channel subsystems. In most cases, a subchannel represents an addressable device. A disk control unit with 30 drives uses 30 subchannels (for base addresses), and so forth. An addressable device is associated with a device number and the device number is commonly (but incorrectly) known as the device address.

Subchannel numbers (including their implied path information to a device) are limited to four hexadecimal digits by architecture. Four hexadecimal digits provide 64 K addresses, known as a set. IBM has reserved 256 subchannels, leaving more than 63 K subchannels for general use.
6.4 CSS configuration management

CSS configuration management
The SAP PU schedules the I/O request towards a configured channel, and then the channel
to the control unit and device. The I/O Configuration Data Set (IOCDS) defines the channels,
control units, and devices to the designated logical partitions (LPs) within the CSS, within the
server. All this is defined using the Input/Output Configuration Program (IOCP).

The IOCP statements are typically built using the Hardware Configuration Dialog (HCD). This
interactive dialog is used to generate the Input/Output Definition File (IODF), invoke the IOCP
program, and subsequently build your Input/Output Configuration Dataset (IOCDS). The
IOCDS is loaded into the Hardware System Area (HSA) and initialized during Power-on
Reset. On a System z, the HSA allocation is controlled via the maximum number of possible
devices, as declared in HCD/IOCP. Refer to “Hardware Configuration Definition (HCD)” on
page 427, for more information.

In System z servers, the channel path identifiers are mapped to Physical Channel Identifiers
(PCHID) via the configuration build process. Refer to “Physical channel ID (PCHID)” on
page 253 for more information.

The following tools are provided to maintain and optimize the I/O configuration of your z9 EC.

Hardware Configuration Definition (HCD)
HCD supplies an interactive dialog to generate your I/O definition file (IODF) and
subsequently your Input/Output Configuration Dataset (IOCDS). It is strongly recommended

Tools that maintain and optimize the I/O configuration of System z processors:

- Hardware configuration definition (HCD)
- IBM configuration for e-business (eConfig)
- Hardware Configuration Manager (HCM)
- IBM System z CHPID Mapping Tool (CMT)
that you use HCD to generate your IODF, as opposed to writing your own IOCP statements. The validation checking that HCD performs as you enter data helps eliminate errors before you implement your I/O configuration. Refer to “What is HCD” on page 492 for more information about this topic.

**IBM Configurator for e-business (eConfig)**
This tool is used by your IBM representative. It is used to configure new server features, or to maintain features on your existing servers.

**Hardware Configuration Manager (HCM)**
The Hardware Configuration Manager (HCM) is a product that uses a graphical interface to create a bridge between your installation and HCD.

**System z CHPID Mapping Tool (CMT)**
The System z CMT provides a mechanism for customizing the CHPID assignments for a System z server. The purpose is to avoid connecting critical paths to single points of failure. Additional enhancements have been built into the CMT to help with the new requirements of the System z server. Refer to “Configuration definition process” on page 259 for more information about this topic.
6.5 Displaying channel types

Displaying channel types

Figure 6-5 shows the D M=CHP command output. The symbol explanations shown on the right side of the figure follow the information shown on the left on the operator console.

This z/OS operator command provides information about the status and type of channels. There are two parts to the display:

1. The first section displays the channel path status. The channel path status is relative to the z/OS where the command is issued. That is, a CHPID may be displayed as being offline, but if this CHPID is shared (MIF) by other logical partitions, it may not be offline physically.

2. The second section displays the channel path type. Note that where the first section only displays the status of channels available to the z/OS image, the second section provides information about all channels installed on the server.

Channel path types

The channel subsystem may contain more than one channel type. Examples of channel path types used by the channel subsystem are ESCON, FICON Express, FICON bridge and FCP. The term “serial-I/O interface” is used to refer to the ESCON I/O interface, the FICON I/O interface, and the FICON-converted I/O interface.
6.6 ESCON architecture

- Fiber optic
- Point-to-point topology
- Communicates from the CECs to peripherals
- Supports multiple path connections to peripherals
- Provides CTC processor-to-processor communication
- Provides routing information
- Detects media connection and disruption
- Identifies and verifies paths

Figure 6-6  ESCON architecture

**Fiber optic**

ESCON channels have a channel-to-control-unit I/O interface that uses optical cables as a transmission medium. Optical technology is less susceptible to errors caused by electrical noise. Optics also have very low radiation properties, which make them more secure than electrical cables. The fiber optic cable is physically a pair of optical fibers that provide two dedicated, unidirectional serial-bit transmission lines. Information in a single optical fiber always flows in the same direction. Thus, one optical fiber is used to receive data, while the other is used to transmit data.

ESCON provides bidirectional (not concurrent) serial-bit transmission, in which the communication protocol is implemented through sequences of special characters and through formatted and architectured defined sets of characters. A *sequence* is a set of characters in a predefined order used to signal specific states or transition to states, such as a port entering offline state. The ESCON I/O interface defines two types of frames, one to control the link and associated elements, and another to control device operation. Each frame contains addressing information that uniquely identifies the sender and the receiver.

An ESCON channel is composed of two parts:

- ESCON channel itself - a type of channel that uses an optical fiber communication link between channels and control units and is in the I/O card in the I/O cage
- ESCON port - a connector in the same I/O card
ESCON Directors (ESCDs) add dynamic switching capability for ESCON channels, further increasing connectivity and device sharing.

**Point-to point topology**
ESCON is essentially a point-to-point (or one channel-to-one control unit) topology that establishes a *serial* protocol and media specification for the communication among channels and control units. The ESCON channel implementation uses fiber optic technology, which is ideally suited for high speed operation over long distances.

**Communication from servers**
ESCON I/O and the interconnect technologies become very important in a sysplex to speed access to shared data on disk or tape, enhancing communication among systems. In addition, these technologies offer improved availability.

**Multiple path connections**
The switching capabilities allow multiple connections between channels and control units (using switches) without requiring static (dedicated) physical connections. The point-to-point connections allow physical changes to the I/O configuration concurrently with normal operations on other ESCON paths. Both topologies are implemented by defining an ESCON channel as CNC.

**ESCON CTC support**
ESCON offers an effective and price-competitive replacement for previous channel-to-channel hardware. With ESCON channels, a user can communicate at channel speed between servers without requiring extra hardware. The ESCON channel-to-channel (CTC) is an IOCP configuration option of an ESCON-capable server. The CTC option is specified in the IOCP configuration, which results in the CTC LIC being loaded into the ESCON channel hardware at power-on Reset (POR).

**ESCON routing information**
With point-to-point topology, the only way of reaching several control units is through routing data using dynamic switches. The ESCON Director, under frame content control, routes transmission streams from one port to any other port in the same ESCON Director, and provides data and command exchange for multiple channels and control units.

**ESCON disruption detection**
With ESCON disruption detection, if one of the activated switch ports fails, the system performs a *call home* to inform IBM. An IBM Service Representative will initiate the repair by selecting the “Perform a Repair Action” icon at the Service task panel on the SE. This will start the Repair & Verify procedure.

**Identifies and verifies paths**
Data is routed to control units using dynamic switches known as 9032 ESCON Directors (ESCD). These directors can be used to connect multiple channels to the same control unit, or multiple control units to the same channel. Also, they allow you to connect channels to other channels (CTCs), and control units to other control units. ESCDs allow longer distances and high flexibility in an I/O reconfiguration, mainly in a failure recovery situation.

The directors can be controlled and managed from host-based programs such as Auto Operator and HCD. z/OS, when communicating with the ESCDs (when changing ports state, for example), uses a specific device number (known as a CUP) and a specific unit address assigned in the ESCD.
EMIF
ESCON Multiple Image Facility (EMIF) allows the same physical channel to be shared among multiple images from an LPAR. Today this feature is known as just MIF, to be used for ESCON and FICON channels.
### 6.7 ESCON concepts

**ESCON concepts**

ESCON and the z/Architecture introduce extensions to the device addressing scheme described previously. However, most of the changes are transparent to the application program. Figure 6-7 shows the ESCON device path structure. The daisy-chained parallel channel connecting to two different control units is shown for comparison. Remember that the old parallel channels are not supported in the System z family of servers.

**ESCON link**

An ESCON channel may connect either directly to an ESCON-capable control unit (which is known as a *point-to-point* connection), or connect to an ESCON-capable control unit through an ESCON Director (known as a *switched* connection). Information in a single optical fiber flows, bit by bit, always in the same direction. The ESCON link data rate is 20 MB/sec. At any link interface, one optical fiber of the pair is used to receive data. The other optical fiber of the pair is used to transmit data. The link is used to attach and interconnect other elements of the ESCON I/O interfaces.

The ESCON environment provides increased connectivity by providing for each ESCON channel to connect to up to 256 links. Each link (port) may attach to a control unit. Figure 6-7 shows one ESCON channel connecting to two different ESCON links, and therefore two different physical control units. One of them is a CTC (connected to another server), and the other refers to a control unit with different control unit images.
Link address
The link address is a two-digit hex number, in the range 01-to-FE, which identifies the sender and the receiver during the communication between an ESCON channel and its control units through 1 KB frames. In other words, link addresses are used in the ESCON frame to identify both destination and source link addresses. Link addresses 00 and FF are reserved by hardware.

A link connects to an ESCON Director through a port. Each port in a ESCD has a port number from 01 to FF.

The link address (when the channel is connected to a ESCD) corresponds to the ESCON Director port number where the ESCON link attaches. If the channel is not connected to an ESCD port, then the link address is FE.

Here is an example. A channel (sender) is starting a conversation with a control unit (receiver), sending an ESCON frame through an ESCD. The receiver address is made up of the following elements:

- A link address associated with the ESCD port number of the control unit (passed by the installation in HCD). This information is used by the ESCD to route a frame originating in the channel port to the appropriate link port where the physical control unit is connected. The link address value is shown as 3A in Figure 6-7 on page 390.

- A control unit image (CUADD value). Some physical control units that support single-path devices, such as the ESCON-capable DS8000 family, allow concurrent connections to multiple hosts through the same physical interface, or control unit adapter. This support is provided through the use of different Control Unit Images (CUI) or logical control units. As with the link address, the CUI, also known as the control unit address (CUADD), forms part of the device selection scheme in an ESCON environment. The control unit image value is shown as 01 in Figure 6-7 on page 390.

- A device unit address.

The sender (or source) address indicates the following:

- A channel link address (port number in the ESCD). This information is not provided by the HCD; instead, the channel obtains this value from the ESCD at initialization. This is shown as 2A in Figure 6-7 on page 390.

- An LP MIF ID to allow the implementation of MIF, shown as 03 in Figure 6-7 on page 390.

When responding to the channel, the control unit swaps the receiver address and the sender address in the new frame.

There are no MVS commands to display the ESCON components of a path to a device. Only Auto Operator (which has the functions of the ESCON Manager product) commands can display the link addresses used by channels supporting the paths to a device.
6.8 ESCD (switch) functions

ESCD (switch) functions

To implement the switching functions of ESCON, a class of product was introduced known as the ESCON Director (ESCD). It dynamically connects channels and control units only for the duration of an I/O operation. The connection is “dynamic” because it is only held during the conversation. After that, the same port can be used to connect to a different control unit or different channel.

ESCDs do not introduce delays, and can switch millions of connections per second. ESCDs are the centerpiece of the ESCON topology.

Switching functions

ESCDs switch connections between ports under the direction of the link address, as provided by the channel and the attached ESCON control units within the frames. Because the ESCD redrives the light signal, it can be used as a channel extender to communicate over long distances.

Apart from dynamic switching, the ESCD can also be used for static switching (also called dedicated), where a port is always connected to another port. When a channel and a control unit are connected through two ESCDs (for longer-distance purposes), one of the connections must be static, because in the frame there is only one sender link address and only one receiver link address.

Switching function terminology

- Prohibited Port
- Allowed Port
- Chained Connection
- Dedicated Port

Figure 6-8  ESCD (switch) functions
To store configurations and handle errors the director has an integrated control unit, which is addressed by the host like any other control unit. The director dynamically switches I/O requests for itself to its own internal port.

**Port connections**

The ESCON Director (ESCD) temporarily connects ports for transmission. The connection can remain in effect for the duration of the I/O operation while several frames are exchanged between the ports. The connection is made (and broken) on request from a frame sent by either the control unit or the channel.

ESCD ports are **non-blocking**, in the sense that there are enough internal paths so that all ports can communicate simultaneously. Signals travel from one port to another, converting from optical signals to electrical and back to optical. This allows for some “elasticity” in the bit stream (source and destination ports do not have to be locked at exactly the same time), and to power the optical signal, balancing for any losses at interfaces or on the fiber cables themselves.

The ESCON Director also provides a control unit function that can be used to control port connectivity. Ports can be dedicated to communicate with only one other port, prohibited from communicating with certain other ports, or blocked from communicating altogether.

An ESCON channel path may pass through two ESCON Directors. However, the path connection through one director must be defined on the director as static; that is, the internal port communication for that link must be dedicated.
6.9 ESCON Director (ESCD) description

ESCON Director (ESCD) description
The ESCD consists of multiple bidirectional ports to which channels and control units may be attached. The ports are capable of any-to-any connection, but the installation may restrict this access by:

- Blocking ports (allowing no access at all)
- Prohibiting port-to-port connections
- Dedicating port-to-port connections
- Allowing connections (opposite of prohibiting)

Such restrictions can be used, for example, to prevent a test z/OS system from accessing the production DASD; however, if a disaster occurs, automation acting upon the port’s state may reverse the situation. HCD uses the Auto Operator product to communicate with ESCON Directors.

There are two types of ESCDs, the 9032 ESCD and the 9033 ESCD, as described here:

- The 9032 ESCD has from 28 to 60 external ports (in four-port increments). Each 9032 port provides for the attachment of any ESCON channel, ESCON extended-distance channel, control unit, 9034 or 9035 ESCON Converter, or another ESCD. The 9032 model 5 may contain the FICON bridge card, which converts FICON protocol to ESCON.
- The 9033 ESCD has from 8 to 16 external ports. Each 9033 port provides for the attachment of any ESCON channel or control unit, 9034 or 9035 ESCON Converter, or another ESCD.
6.10 ESCON Director matrix

ESCON switch matrix

The port configuration is held in a switch port matrix in the ESCON Director. The port matrix can be read and written from an attached host using the Auto Operator or the ESCD console. An initial port matrix is held on the disk of a PC directly attached to the director; however, this disk data is not directly addressable by the ESCON Manager and therefore cannot be changed remotely.

Figure 6-10 shows an example of an ESCON switch matrix. The ESCON switch matrix can be stored in a number of places:

- The active switch data is in the ESCD itself.
- The switch may have a copy of the active configuration on its support console disks.
- The Auto Operator may have a copy of the active configuration in its working storage, being worked on as an ISPF table.
- The Auto Operator may have saved copies of possible configurations in ISPF tables.
- HCD may have configurations stored in the active production IODF.
- HCD may have configurations stored in other production IODFs.
- HCD may have configurations stored in work IODFs.

The recommendation for matrix configuration is to protect all ports and only allow access where required. This simplifies I/O problem determination and reduces overhead in event notification.
6.11 Channel-to-channel adapter

In a sysplex environment, the movement of data between z/OS systems through XCF is very important. The classical equipment to implement such movement is called a channel-to-channel adapter (CTCA). Logically a CTCA (sometimes shortened to CTC) is composed of three connected pieces:

- A channel connected to an LP
- An special control unit
- A channel connected to another LP (may be in the same server as the previously mentioned channel)

CTC operation

The CTC works as follows:

1. A program running under z/OS in LP1 wants to move data to its counterpart in z/OS in LP2.
2. This program prepares a channel program (a set of CCWs), for writing 4 KB of data from address 8-M in the memory of LP1.
3. z/OS (IOS code) in LP1 issues the SSCH instruction toward a specific device address (as defined in HCD/IOCP).
4. The channel serving LP 1 is selected by SAP and tries to communicate with the control unit.
5. The control unit disconnects this channel and sends a signal to the channel in LP2, forcing an attention I/O interrupt in z/OS from LP1. With the interrupt is passed a device address (different from the one in the LP1).

6. z/OS, based on the device sending the interrupt, discovers through a Sense CCW that there is a pending write. zOS then passes the control to the counterpart program in LP2, posting it and informing about the pending write.

7. This program in LP2 prepares a channel program (a set of CCWs) for reading 4 KB of data to address 12 MB in the memory of LP2.

8. z/OS (IOS code) issues the SSCH instruction in a specific device address (this may be different from the one in LP1).

9. A channel serving LP 2 is selected by SAP, and it tries to communicate with the control unit.

10. The control unit reconnects the channel from LP1 and the data transfers starts from address 8-M in LP1 to address 12_M in LP2, both channels connected, until the full 4 KB are transferred.

11. Both channels generate I/O interrupts to both z/OSs.
6.12 ESCON CTC support

**ESCON CTCs**

ESCON CTCs are used to allow inter-processor communications (connecting central storage of one z/OS image to central storage of another z/OS image). This communication can be among different LPs in the same server, or among different servers (locally or at long distances).

In ESCON CTC architecture, there is no need for a physical control unit. An ESCON CTC channel is a standard ESCON channel that is defined in HCD as TYPE=CTC. During the initialization of the CHPID, the microcode is loaded into the channel to enable it to support the CTC control unit function on top of the ESCON channel function.

Then, an ESCON CTCA channel path has an ESCON CTCA connection at one end, and either an ESCON (CNC) or a FICON (FCV) channel (that is, a FICON channel converted to ESCON) connection at the other end. To enable any two servers to communicate through an ESCON channel path, the ESCON CTC connection can be at either server.

In order for each LP to communicate with another LP (in the same or distinct server) through a shared (MIF) ESCON CTC channel path, you must specify the logical address of the ESCON CTC control unit. The logical address used has the MIF ID, as defined in HCD/IOCP of your LP.
ESCON CTC communication
As shown in Figure 6-12 on page 398, an ESCON CTC channel must communicate with an ESCON CNC channel; that is, a standard ESCON channel.

Note that an ESCON CNC channel communicating with an ESCON CTC channel can also support other ESCON control units, such as DASD, through the ESCD. An ESCON CTC can also be connected point-to-point to an ESCON CNC channel on another server.

Each ESCON CTC fiber can support up to 512 independent device addresses, each one allocated by a distinct application, but sharing the same physical link.

Point-to-point CTC connection example
Consider Figure 6-12 on page 398 again, but this time with some CHPID and device addresses. Imagine an I/O operation running through the CTC path represented by the CHPID 2A from SYS1 and CHPID 31 from SYS2. An application in SYS1 wants to send data (write) to its peer application in SYS2, which wants to receive such data (read).

Such an operation consists of the following steps:
1. IOS in SYS1 issues a SSCH in CHPID 2A—the channel program has a Write command. The data transfer is halted.
2. IOS in SYS2 receives an I/O interrupt (named attention) in CHPID 31.
3. IOS in SYS2 issues a SSCH in CHPID 31—the channel program has a Sense command. The response for that is the information received by IOS in SYS2, that CHPID 2A is issuing a Write command.
4. IOS in SYS2 issues a SSCH in CHPID 31—the channel program has a Read command. The data transfers starts.
6.13 FICON channels

FICON channel protocol (FC-SB-2) is based on Industry Standard Fibre Channel Architecture from ANSI. FICON coexists with ESCON and relieves several limitations, such as:

- ESCON limitations in bandwidth, in the number of devices and number of control units
- The S/390 architecture limit of 256 channels per server. FICON provides relief because you need fewer channels than ESCON for the same I/O workload. However, in the z9 EC model servers, the limit is 1024 channels per channel subsystem (CSS).
- FICON supplements, but does not replace, ESCON. FICON channels use the same fiber optic cables as ESCON channels.

A FICON channel can be native (FC) or bridge (FCV). With bridge, the FICON channel is connected to an ESCD model 5, which converts the FICON protocol to ESCON protocol. This conversion is done to reach control units with ESCON protocol, through FICON channels.

Figure 6-13 shows that the FICON FCV has a CHPID of FC and connects to switch port number C4, which connects to eight control unit ports. One good reason to have FCVs is to save CHPIDs. The FICON channel requires a FICON interface at the control unit, or needs to be connected to a fiber channel switch port in a switch.
**FICON bridge**
The FICON bridge topology, as we saw, is intended to help provide investment protection for currently installed ESCON control units. The IBM 9032 Model 005 ESCON Director is the only director that supports long wavelength FICON links through the use of a FICON bridge (one port) feature.

One FICON bridge port feature provides connectivity to one FICON LX link. Up to 16 FICON Bridge port features can be installed on a single IBM 9032 Model 005. Current IBM 9032 Model 005 ESCON Directors are field-upgradable to support the FICON bridge port feature, which can coexist with the original ESCON port features in the same director.

FICON Express2 does not support an FCV (bridge) type of FICON topology.

**FICON native (not through a bridge)**
The transmission medium for the FICON interface is a fiber optic cable. Physically, it is a pair of optical fibers that provide two dedicated, unidirectional, serial-bit transmission lines. Information in a single optical fiber flows, bit by bit, in one direction. At any link interface, one optical fiber is used to receive data, while the other is used to transmit data. Full duplex capabilities are exploited for data transfer (not true for ESCON). The Fibre Channel Standard (FCS) protocol specifies that for normal I/O operations, frames flow serially in both directions, allowing several concurrent read and write I/O operations on the same link. There are three types of FICON in System z servers: FICON (not supported by the z9 EC), FICON Express, and FICON Express2.

**FICON distance and capacity**
There are two types of FICON adapters, the long wavelength (LX) and short wavelength (SX), depending on the transceiver. The long wave is for large distances. FICON allows a distance (between server and control unit) of up to 10 km or 29 km (RPQ) with no repeater; or distances of up to 100 km with a repeater.

Each FICON channel allows up to 16,384 devices and up to 256 CUs (one-byte CUADD) with significantly fewer connections, as fewer channels, ports, cables, patch panel ports, and so on are required.
6.14 FICON conversion mode

FICON modes and topologies
System z servers allow FICON channel to operate in one of three modes:
- FICON conversion mode (FCV)
- FICON native mode (FC)
- Fibre Channel Protocol (FCP)

FICON conversion mode
A FICON channel in FICON conversion mode (CHPID type FCV) can access ESCON control units through a FICON Bridge port feature installed in an IBM 9032 Model 005 ESCON Director, as shown in Figure 6-14.

One FICON Bridge port feature provides connectivity to one FICON or FICON Express LX link. Up to 16 FICON Bridge port features can be installed on a single IBM 9032 Model 005. FICON Bridge port feature can coexist with the original ESCON port features in the same Director.

FICON Bridge feature
The FICON Bridge port feature occupies one I/O slot in the IBM 9032 Model 5 cage. Each feature has one FICON LX Bridge port, allowing simultaneous connections to 8 different control units on eight different ESCON links. On average, one FCV may be equivalent
(performance-wise) to up to four ESCON channels, depending on the utilization of the eventually replaced ESCON channels.

Up to 16 FICON Bridge port features can be installed in a FICON-capable IBM 9032 Model 5. These features plug into ESCON port feature locations in the director, reducing the number of I/O slots available for ESCON ports.

The maximum FICON Bridge port feature configuration allows:

- 16 FICON LX Bridge ports (16 FICON Bridge port features)
- 120 ESCON ports (15 ESCON 8-port features)

**Note:** With FICON channels (in FCV mode), you can still preserve your ESCON control unit investments.

**I/O operation**

An I/O operation is “FICON FCV mode-transferred” from the FICON channel to the FICON Bridge port. The FICON Bridge port translates FC-SB-2 (INCITS standard) frames into ESCON frames, and conversely, ESCON frames into FC-SB-2 frames. The channel side of the FICON bridge port operates in a slightly modified FCS mode (full duplex). The ESCON side operates in normal ESCON mode (half duplex).
6.15 Supported FICON native topologies

FICON native mode
A FICON channel in FICON native mode (CHPID type FC) can access FICON native interface control units using the topologies shown in Figure 6-15, as explained here:

- Point-to-point (direct connection)
- Switched point-to-point (via a Fibre Channel Director)
- Cascaded FICON Directors (through two Fibre Channel Directors)

A FICON native channel also supports channel-to-channel communications. The FICON channel at each end of the FICON CTC connection, supporting the FCTC control units, can also communicate with other FICON native control units, such as disk storage devices and tape.

Note that at least one end of the FICON CTC connection must be System z servers. In FCTC, there is no need to define one of the channels as a CTC channel in HCD.
6.16 Fibre Channel Protocol (FCP)

Fibre Channel Protocol mode (FCP)
This channel is not supported by z/OS. A FICON channel in Fibre Channel Protocol mode (CHPID type FCP) can access FCP devices either through a single:

- Fibre Channel switch or multiple switches to an FCP device
- Fibre Channel switch or multiple switches to a Fibre Channel-to-SCSI bridge

The System z server FICON Express and FICON features provide support for Fibre Channel and Small Computer System Interface (SCSI) devices in Linux environments. This support is in conjunction with the Linux distributions from System z Linux distribution partners.

Note: zSeries FCP channel direct attachments in point-to-point or arbitrated loop topologies are not supported as part of the zSeries FCP enablement.

The zSeries FCP support allows Linux running on a zSeries server to access industry-standard SCSI devices. For disk applications, these FCP storage devices utilize Fixed Block (512-byte) sectors rather than Extended Count Key Data (ECKD™) format.

FCP and SCSI controllers and devices can be accessed by Linux for zSeries (64-bit mode) or Linux for S/390 (31-bit mode) with the appropriate I/O driver support.

Linux may run either natively in an LP, or as a guest operating system under z/VM Version 4 Release 3 and later releases, to support FCP for Linux guests.
6.17 FICON improvements (1)

FICON versus ESCON
The best way to understand a FICON channel facilities is to compare them with the ones in ESCON; we assume that you are already familiar with ESCON concepts.

We can draw an analogy between this protocol comparison and two people talking on the phone, as pictured in Figure 6-17 and in Figure 6-18 on page 408:

- ESCON protocol is like having two people (channel and controller) talking on the phone about just one I/O operation; when one person talks, the other person listens.
- FICON protocol is like having two sets of people (channels and controllers) talking on phones connected to a switchboard, about several subjects (I/O operations). Each person is talking and each person is buffering what the other is saying for further analysis. In this conversation, one person from one group may talk to more than one person of the other group, concurrently.

Comparison conclusions
FICON provides all the strengths of ESCON, while increasing the link rate from 20 MB/sec all the way up to 100 MB/sec on the FICON feature, and up to 200 MB/sec on the FICON Express features. It also increases the extended (non-droop) distance from 9 km to 100 km.

Extended Distance FICON is an enhancement to the industry standard FICON architecture (FC-SB-3) helps avoid degradation of performance at extended distances by implementing a new protocol for “persistent” Information Unit - IU (also named frames) pacing.
Control units that exploit the enhancement to the architecture can increase the pacing count (the number of IUs allowed to be in flight from channel to control unit). Extended Distance FICON also allows, for example, keep a 400 MB/sec link fully utilized at 50 km.

The FICON implementation enables full duplex data transfer, so data travels in both directions simultaneously, rather than the half-duplex data transfer of the ESCON implementation. Also, FICON enables multiple concurrent I/O operations, rather than the single-sequential I/O operation of ESCON.

Comparing both methods, we can reach the following conclusions:

- Throughput is better in FICON protocol because it is full duplex, asynchronous (no waits for the acknowledgements of CCWs and data), multiplexes different I/O requests, and requires significantly fewer handshakes (fewer acknowledgements).
- Distance has less effect on throughput in FICON protocol, because there is no waiting for acknowledgements.
- Higher throughput in FICON protocol means less I/O operation queue time.
- Service time per I/O operation is not clear: FICON protocol does not need synchronous handshakes, but ESCON protocol is completely dedicated to the I/O operation.
- Recovery time in ESCON protocol, if needed, is shorter.

**FICON provides expanded addressability**

Many of the ESCON addressability constraints are relieved by FICON, as described in 6.19, “FICON/ESCON numerical comparison” on page 410.
6.18 FICON improvements (2)

Higher data rate
A higher data rate (20 MB/sec for direct and 70 MB/sec for sequential) produces better performance and better channel consolidation (fewer channels in the configuration). Note that the nominal bandwidth of a FICON Express channel is 200 MB/sec. The reasons for such a higher data rate are:

- FICON implementation enables full duplex data transfer, so data frames with 128 KB travel in both directions simultaneously, rather than the ESCON half-duplex data transfer.

  Also, multiple concurrent I/Os (up to 64 per FICON Express2 in the z9 EC) can occur on a single FICON channel, where consecutive data frames may belong to distinct I/O operations. Each concurrent I/O is called *open exchange*. This is a fundamental difference between FICON and ESCON.

- FICON has asynchronous command execution and consequently, multiple concurrent I/Os. It implements CCW Pipelining, where all CCWs are sent without waiting for channel end (CE)/device end (DE). In other words, CCWs (CCW1-CCWn) are transferred to the controllers without waiting for the first command response (CMR) from the controller or for a CE/DE after each CCW execution. Although the device presents a DE to the control unit after each CCW execution, for the last CCW of the chain, the control unit presents CE/DE to the channel.

- FICON has asynchronous data transfer. The channel sends multiple data frames of the same I/O operation along a write, without being delayed by acknowledgement signals from the controller (and vice versa, for reads).
FICON channels are almost always available. When migrating from ESCON to FICON, you may see a decrease in Pending and Disconnect time and possibly some increase in connect time, due to an internal multiplex or queue. However, in general, mainly in large block sequential access, the net effect is faster I/O response time with FICON.

The disconnect and reconnect protocols are eliminated between FICON channels and FICON Host adapter controllers. When read cache missed data is finally in the cache, the controller places the data frame in the output queue, waiting to be picked by the channel polling mechanism; there is no reconnect mechanism.

There are no port busy handshakes like the ones caused by the ESCON Director port busy. Collisions within a FICON switch are completely eliminated. Data frames are intermixed in the FICON switch ports. However, there is still internal FICON switch queue time if the port becomes overloaded. This time is reported as pending time, but not under DP Busy Delay in RMF.

However, because several I/Os are processed at the same time, there may be some increase in the I/O internal queue time (shown in connect time). There is also no guarantee that the I/O response time for a direct I/O (short block) always decreases when migrating from ESCON to FICON. However, the throughput is much better. FICON channels experience minimal data rate droop at distances up to 100 km (62 miles). The reasons are:

- Increased buffer sizes in the FICON channel card.
- There is only one handshake, due to the asynchronous data transfers and CCW execution in FICON. There is only a propagation delay of 0.1 microseconds at each 10 km in FICON, against 0.6 microseconds in ESCON.

FICON features

There are three types of FICON features available in System z servers:

- **FICON** - With a bandwidth of 100 MB/sec full duplex, with 60 to 80 MB/sec expected total data rate, depending on the type of workload, fiber infrastructure, and storage devices in place. FICON can deliver more than 4,000 I/Os/sec (4 KB blocks) per channel, compared to 500 I/Os/sec (4 KB blocks) per channel in ESCON.

- **FICON Express** - With a bandwidth of 100 MB/sec or 200 MB/sec, delivering from 20% to 30% real improvement over FICON. Note that the control unit adapter (also called the host adapter) needs to be able to communicate with a FICON Express channel. The link speed is auto-negotiated, point-to-point, between channel and controller; it is transparent to users and applications. Not all controllers support 200 MB/sec link data rates.

- **FICON Express2** - Supported by z9 EC, z990, or z890 servers. Allows installing twice the number of FICON channels in the same space while benefiting from increased performance, as compared with the previous generation of FICON Express features. The FICON Express2 SX and LX features have four independent ports, each feature occupying a single I/O slot, utilizing four CHPIDs per feature, while continuing to support 100 MB/sec and 200 MB/sec link data rates. The FICON Express2 SX and LX features are ordered in 4-port increments and designed to be added concurrently. This concurrent update capability allows you to continue to run workloads through other channels while the FICON Express2 features are being added. FICON Express2 (CHPID types FC and FCP) can be defined as a spanned channel and consequently can be shared among LPs within and across CSSs. FICON Express2 SX and LX features replace the current FICON Express SX and LX features currently offered on z990 and z990. On the z9 EC model, we may have 84 FICON features (all I/O slots in the three I/O cages) with 4 x 84 = 336 FICON Express2 channels.
6.19 FICON/ESCON numerical comparison

FICON surpasses ESCON in the following numerics:

- Control Unit images per physical control units
- Unit addresses per channel
- Unit addresses per physical control unit
- Logical paths per port (in ESS)
- Number of concurrent I/O operations
- Link data rate
- Channel frame transfer buffers
- Max distance

Figure 6-19  FICON/ESCON numerical comparison

FICON/ESCON numerical comparison

Table 6-3 on page 411, and Table 6-4 on page 411 and Table 6-2, portray numerically some of the differences between FICON and ESCON channels.

Concurrent operations, data rates, transfer buffers, and distance

Table 6-2  ESCON and FICON characteristics

<table>
<thead>
<tr>
<th></th>
<th>ESCON channels</th>
<th>FICON channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command execution</td>
<td>Synchronous to CU</td>
<td>Asynchronous to CU</td>
</tr>
<tr>
<td>Channel concurrent I/O operations</td>
<td>1</td>
<td>Up to 32</td>
</tr>
<tr>
<td>Link data rate</td>
<td>20 MB/ps</td>
<td>Up to 200 MB/ps</td>
</tr>
<tr>
<td>Channel frame transfer buffers</td>
<td>1 KBytes</td>
<td>128 KBytes</td>
</tr>
<tr>
<td>Max distance without repeat</td>
<td>3 km</td>
<td>10 km</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 km at 200 MB/ps RPQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 km at 100 MB/ps RPQ</td>
</tr>
<tr>
<td>Max distance without droop</td>
<td>9 km</td>
<td>100 km</td>
</tr>
</tbody>
</table>
Control units and addresses/channel

Table 6-3  ESCON and FICON channel - maximum addressing

<table>
<thead>
<tr>
<th></th>
<th>ESCON channels</th>
<th>FICON channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>CU images (CUADD) / CU:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architected</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>Implemented</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>Unit Addresses / channel:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architected</td>
<td>1 M</td>
<td>16 M</td>
</tr>
<tr>
<td>Implemented</td>
<td>1 K</td>
<td>16 K</td>
</tr>
<tr>
<td>UAs/physical CU:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architected</td>
<td>4 K</td>
<td>64 K</td>
</tr>
<tr>
<td>CU-implemented</td>
<td>4 K</td>
<td>4 K</td>
</tr>
<tr>
<td>Addressable by a channel</td>
<td>1 K</td>
<td>4 K</td>
</tr>
<tr>
<td>UAs/logical CU (CUADD)</td>
<td>256</td>
<td>256</td>
</tr>
</tbody>
</table>

Logical paths in Enterprise Storage Server (ESS)

A logical path is a control block within the control units, where each instance of a channel is registered. Each MIU image of a channel consumes one logical path. Table 6-4 lists the addressing attributes in IBM Enterprise Storage Server® (ESS), ESCON channels, and FICON channels.

Table 6-4  Control Unit addressing attributes

<table>
<thead>
<tr>
<th>IBM Enterprise Storage Server Model 800</th>
<th>ESCON channels</th>
<th>FICON channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical paths/per port</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>Logical paths/logical control unit</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>
6.20 FICON switches

FICON channels implement the switched point-to-point topology by using FICON switches to connect FICON channels to control unit FICON adapters. These connections are established as the server FICON channel discovers that there is a port switch between it and the control unit. All FICON switch connections are dynamic. So static connections, which are possible in an ESCON Director, are not supported in a FICON Director.

**FICON/Fibre channel switch**

The fibre channel switch (FC-SW) supports *packet-switching*, which provides better utilization than the *circuit-switching* in the ESCON Director. It allows up to 32 simultaneous concurrent I/O operations (read and write) from multiple FICON-capable systems to multiple FICON control units.

Figure 6-20 shows a conceptual view of frame processing in a switched point-to-point configuration for multi-system and multi-control unit environments.
6.21 Cascaded FICON Directors

Figure 6-21 Cascaded FICON Directors with FICON switched fabric at two sites

FICON switched fabric
An important point to note regarding the FICON fabric switch is that there are no dedicated links between a port in the first switch with a port in the second switch. All data transfer is done through a common set of links named Inter-switch links (ISL). The result of this design is better performance, and there is no need for static connections.

Cascaded FICON Directors
FICON native and FICON Express channels on System z servers support cascaded Fibre Channel Directors. This support is for a two-Director configuration only. With cascading, a FICON native channel, or a FICON native channel with channel-to-channel (FCTC) function, can connect a server to a control unit or other server via two native connected Fibre Channel Directors. Two director cascading requires a single vendor high integrity fabric. Directors must be from the same vendor since cascaded architecture implementations can be unique.

FICON channels require z/OS V1R3 with PTFs (or later releases) to enable cascaded FICON Directors. In addition, z/VM V4R4 provides this support. FICON support of cascaded Directors is sometimes referred to as cascaded switching or 2-switch cascaded fabric.

This type of cascaded support is important for disaster recovery and business continuity solutions because it can help provide high availability, extended distance connectivity, and (particularly with the implementation of 2 Gb/sec Inter Switch Links) has the potential for fiber infrastructure cost savings by reducing the number of channels for interconnecting the two sites.
FICON two-Director cascaded technology can allow for shared links, and therefore improved utilization of intersite connected resources and infrastructure. Solutions such as Geographically Dispersed Parallel Sysplex (GDPS) can benefit from the reduced inter-site configuration complexity that Native FICON support of cascaded Directors provide. Refer to ABCs of z/OS System Programming Volume 9 for more information on GDPS.

Data centers between two sites
While specific cost savings vary depending upon infrastructure, workloads, and size of data transfers, generally customers who have data centers separated between two sites may reduce the number of cross-site connections by using cascaded Directors. Further savings may be realized in the reduction of the number of channels and switch ports.

Another advantage of FICON support of cascaded Directors is its ability to provide high integrity data paths. The high integrity function is an integral component of the FICON architecture when configuring FICON channel paths through a cascaded fabric.

Data integrity
End-to-end data integrity is designed to be maintained through the cascaded Director fabric. Data integrity helps ensure that any changes to the customer's data streams are always detected, and the data frames (data streams) are delivered to the correct end point, an end point being a FICON channel port or a FICON Control Unit port.

For FICON channels, Cyclical Redundancy Checking (CRC) and Longitudinal Redundancy Checking (LRC) are bit patterns added to the customer data streams to allow for detection of any bit changes in the data stream.

With FICON support of cascaded switching, new integrity features are introduced within the FICON channel and the FICON cascaded switch fabric to help ensure the detection and reporting of any miscabling actions occurring within the fabric during operational use that may cause a frame to be delivered to the wrong end point.
6.22 FICON Channel to Channel Adapter (FCTC)

FICON FCTC support

Here are major differences between the ESCON and FICON CTC implementation:

- ESCON CTC communication requires a pair of ESCON channels, where one of the channels is dedicated to the ESCON CTC function and is defined as an ESCON CTC channel, and the other channel of the pair is defined as a normal ESCON CNC channel. For FICON, the FCTC function does not require a FICON channel to be dedicated to support the FCTC function; only a normal FICON channel defined in FICON (FC) mode channel is required. The same channel can also be used to support other I/O device-type operations.
- The FICON FCTC communication can use either one FICON channel (only recommended for a small number of images in the CTC complex), or two FICON channels. In the case of one FICON channel, the connected LPs must reside in the same server.
- The number of unit address associated to an ESCON CTCS is 512; in an FCTC, it is 16 K.
- Data transfer bandwidth in the ESCON CTC is 12 - 17 MB/sec; in a FCTC, it is 60 - 90+ MB/sec.
- The number of concurrent I/O operations is one in the ESCON CTC; it up to 32 in FCTC.

As shown in Figure 6-22, the same FICON channel from Server 1 is doing the following:

- Establishing an FCTC connecting with LP1, LP2 and LP3
- Accessing the DASD control unit
- Establishing an FCTC connecting the LPs from Server 1 with Server 2
  - FCTC supports cascade switches
Parallel Sysplex technology is a highly advanced, clustered, commercial processing system. It supports high performance, multisystem, read/write data sharing, enabling the aggregate capacity of multiple z/OS systems to be applied against common workloads. A Parallel Sysplex comprises one or more z/OS systems coupled through one or more Coupling Facilities.

The z/OS systems in a Parallel Sysplex configuration have all the capabilities of the standard z/OS system z system, and run the same applications. This allows users to harness the power of multiple z/OS systems, as if they were a single, logical computing system. Its major appeal is continuous availability (24x7), where all z/OSs are cloned (because of data sharing) and consequently have no single points of failure (SPOF) along planned or unplanned outages.

Coupling Facility
The Parallel Sysplex technology is centered around the implementation of a Coupling Facility, an LP running the coupling facility control code (CFCC) operating system and high speed coupling connections for inter-server and intra-server communications. The Coupling Facility is responsible for providing high speed data sharing with data integrity across multiple z/OS systems.
Coupling Facility (CF) links
The type and mode of CF links you can use to connect a CF to an operating system LP is important. The type of coupling links used to connect a CF to an operating system logical partition is important, because of the impact of the link performance on response times and coupling overheads. For configurations covering large distances, the time spent on the link can be the largest part of the response time.

There are two types of modes: old non-peer mode (also called sender/receiver), and new peer mode.

Peer mode links
There are several advantages in using peer mode links:

- All peer mode links operate on a higher speed than the equivalent non-peer mode link.
- A single CHPID (one side of the link) can be both sender and receiver; this means that a single CHPID can be shared between multiple OS LPs and one CF LP.
- The number of subchannels defined when peer mode links are used is seven (7) per LP per link compared to two (2) per LP per link in compatibility mode. This is particularly important with System-Managed CF Structure Duplexing.
- Peer links have 224 KB data buffer space compared to 8 KB on non-peer links; this is especially important for long distances as it reduces the handshaking for large data transfers.

The CF links in peer mode are the ones with a suffix of -3 or a -4 in the type name, as shown in Figure 6-22 on page 415. With the exception of the IC channel, all the other channels have a corresponding PCHID.

When z9 and z10 EC servers are connected through CF links, the mode is peer mode only.

Types of CF links
The following types of links are available to connect an operating system LP to a CF on a System z:

- **IC** - A microcode-defined link to connect a CF LP to a z/OS LP in the same System z server. IC links require two CHPIDs to be defined, and can only be defined in peer mode. The link speed is greater than 2 GB/sec. A maximum of 32 IC links can be defined.
- **ICB-4** - A copper link available to connect the z/OS LP and the CF LP (in different servers). The maximum distance between the two servers is 7 meters (the maximum cable length is 10 meters). The link speed is 2 GB/sec. ICB4 links can only be defined in peer mode. The maximum number of ICB4 links is 16. They are directly connected to the MBA in the server cage through a dedicated STI (in a z9) and a dedicated IFB (in a z10 EC). This link supports transmission of server time protocol (STP) messages.
- **ISC-3** - A fiber link defined in peer mode available to connect System z servers (CF LP and z/OS LP in different servers with a distance of more than 7 meters). The maximum distances are: 10 km; 20 km with an RPQ; or 100 km with dense wave division multiplexing (DWDM) connections. The link speed is 200 MB/sec (for distances up to 10 km), and 100 MB/sec for greater distances (RPQ). The maximum number of ISC links is 48. These links are located in a special set of I/O cards named “mother” and “daughters”. The System z ISC-3 mother card occupies one slot in the I/O cage. The ISC-3 mother card supports up to two ISC-3 daughter cards. Each ISC-3 daughter card has two independent ports, with one PCHID associated with each active port. ISC-3 supports transmission of STP messages.
6.24 z10 EC Coupling Facility connectivity options

In addition to the connectivity options described in “z9 Coupling Facility links” on page 416, in z10 EC the following options are available:

- **Parallel Sysplex using InfiniBand (PSIFB)** - Fiber connects a z10 EC to another z10 EC, or a z10 EC to a standalone CF on a z9 EC or z9 BC. It supports a maximum distance of 150 meters. PSIFB coupling links are defined as CHPID type CIB, and operate at 6 GB/sec when connected to a z10 EC, or at 3 GB/sec when connected to z9 standalone CF. The maximum number of PSIFB links is 32 per z10 EC. PSIFBs support transmission of STP messages. Refer to “Connecting PU cage with I/O cages” on page 149.

- **z10 EC has Host Channel Adapter (HCA) fanouts residing on the front of the book.** There are unique HCA fanouts supporting I/O features and Parallel Sysplex coupling links in the I/O cage. The HCA2-O fanout provides an optical interface used for coupling links. The two ports on the fanout are dedicated to coupling links to connect to z10 EC or z9 servers, or to connect to a coupling port in the same server using a fiber cable. Each fanout has an optical transmitter and receiver module, and allows dual simplex operation. Up to 16 HCA2-O fanouts per book are supported and provide up to 32 ports for coupling links.

- **InfiniBand coupling links offer up to 6 GB per second of bandwidth between z10 EC servers and up to 3 GBps of bandwidth between System z10 and z9.** InfiniBand coupling links (CHPID type CIB) are designed to complement and enhance the support provided by traditional coupling link technology (ICB-4 and ISC-3). InfiniBand coupling links can be used to carry Server Time Protocol (STP) messages.
6.25 All z10 EC coupling link options

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Use</th>
<th>Link rate</th>
<th>Distance</th>
<th>z10 EC maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISC-3</td>
<td>Fiber connection</td>
<td>z10 EC to z10 EC, z9 EC, z9 BC, z990, z890</td>
<td>2Gbps</td>
<td>10 km un/repeated (6.2 miles) 100 km repeated</td>
<td>48</td>
</tr>
<tr>
<td>ICB-4</td>
<td>Copper connection</td>
<td>z10 EC to z10 EC, z9 EC, z9 BC, z990, z590</td>
<td>2Gbps</td>
<td>10 meters (33 feet)</td>
<td>16</td>
</tr>
<tr>
<td>PSIFB</td>
<td>12x IB-DDR Fiber connection</td>
<td>z10 EC to z10 EC, z9 EC, z9 BC</td>
<td>6Gbps</td>
<td>150 meters (492 feet)</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>12x IB-SDR</td>
<td>Internal Coupling Channel</td>
<td>Internal communication</td>
<td>3Gbps</td>
<td>16</td>
</tr>
<tr>
<td>IC</td>
<td>Internal Coupling Channel</td>
<td>Internal communication</td>
<td>Internal speeds</td>
<td>N/A</td>
<td>32</td>
</tr>
</tbody>
</table>

Figure 6-25  All z10 EC coupling link options

All z10 EC coupling link options

Figure 6-25 shows all the possibilities for connecting z10 EC Coupling Facility links. Coupling links are required in a Parallel Sysplex configuration to provide connectivity from the z/OS images to the Coupling Facility. A properly configured Parallel Sysplex provides a highly reliable, redundant, and robust System z technology solution to achieve near-continuous availability. A Parallel Sysplex comprises z/OS operating system images coupled through one or more Coupling Facilities.

The type of coupling links used to connect a CF to an operating system logical partition is important because of the impact of the link performance on response times and coupling overheads. For configurations covering large distances, the time spent on the link can be the largest part of the response time.

The types of links that are available to connect an operating system logical partition to a Coupling Facility are:

**ISC-3** The ISC-3 feature is available in peer mode only. ISC-3 links can be used to connect to z10 EC, z9 EC, z9 BC, z990, or z890 servers. They are fiber links that support a maximum distance of 10 km, 20 km with RPQ 8P2197, and 100 km with Dense Wave Division Multiplexing (DWDM). ISC-3s operate in single mode only. Link bandwidth is 200 MBps for distances up to 10 km, and 100 MBps when RPQ 8P2197 is installed. Each port operates at 2 Gbps. Ports are ordered in increments of one. The maximum number of ISC-3 links per z10 EC is 48. ISC-3 supports transmission of STP messages.
ICB-4  Connects a System z10 to a z10, z9 EC, z9 BC, z990, or z890. The maximum distance between the two servers is seven meters (maximum cable length is 10 meters). The link bandwidth is 2 GBps. ICB-4 links can only be defined in peer mode. The maximum number of ICB-4 links is 16 per z10 EC. ICB-4 supports transmission of STP messages.

PSIFB  Parallel Sysplex using InfiniBand connects a System z10 to another z10 or a System z10 to a standalone CF on z9 EC or z9 BC. PSIFB links are fiber connections that support a maximum distance of up to 150 meters. PSIFB coupling links are defined as CHPID type CIB. The maximum number of PSIFB links is 32 per z10. PSIFBs supports transmission of STP messages.

IC  Licensed Internal Code-defined links to connect a CF to a z/OS logical partition in the same z10 EC. IC links require two CHPIDs to be defined and can only be defined in peer mode. The link bandwidth is greater than 2 GBps. A maximum of 32 IC links can be defined.
6.26 OSA-Express

OSA-Express features provide significant enhancements over OSA-2 in function, connectivity, bandwidth, data throughput. OSA-Express is the integration of a channel and a telecommunication control unit in an I/O card within an I/O cage. This integration makes the OSA-Express a unique channel, recognized by the hardware I/O configuration as one of the following channel types:

- OSD (Queued Direct I/O), refer to 6.27, “QDIO architecture” on page 423 for more information.
- OSE (non-Queued Direct I/O.)

The integrated control unit depends on the type of connected network.

There are two types of OSA-Express in z9 and z10 EC servers:

- **OSA-Express2**
  - OSA-Express2 Gigabit Ethernet (GbE) Long Wavelength (LX), feature code 3364
  - OSA-Express2 Gigabit Ethernet (GbE) Short Wavelength (SX), feature code 3365
  - OSA-Express2 1000BASE-T Ethernet, feature code 3366
  - OSA-Express2 Gigabit Ethernet 10 GbE LR, feature code 3368

- **OSA-Express3**
  - The OSA-Express3 10 GbE LR
Each feature is explained in more detail in the following sections.

**OSA-Express2 GbE LX (FC 3364)**
This feature occupies one slot in an I/O cage and has two independent ports, with one PCHID associated with each port. Each port supports a connection to a 1 Gbps Ethernet LAN through a 9-micron single-mode fiber optic cable terminated with an LC Duplex connector. This feature utilizes a long wavelength (LX) laser as the optical transceiver.

**OSA-Express2 GbE SX (FC 3365)**
This feature occupies one slot in an I/O cage and has two independent ports, with one PCHID associated with each port. Each port supports a connection to a 1 Gbps Ethernet LAN through a 62.5-micron or 50-micron multimode fiber optic cable terminated with an LC Duplex connector. The feature utilizes a short wavelength (SX) laser as the optical transceiver.

**OSA-Express2 1000BASET-EN (FC 3366)**
This feature occupies one slot in the I/O cage and has two independent ports, with one PCHID associated with each port. Each port supports connection to either a 1000BASE-T (1000 Mbps), 100BASE-TX (100 Mbps), or 10BASE-T (10 Mbps) Ethernet LAN. The LAN must conform either to the IEEE 802.3 (ISO/IEC 8802.3) standard or the DIX V2 specifications.

**OSA-Express2 10 GbE LR (FC 3368)**
This feature occupies one slot in an I/O cage and has one port that connects to a 10 Gbps Ethernet LAN through a 9-micron single-mode fiber optic cable terminated with an LC Duplex connector. The feature supports an unrepeated maximum distance of 10 km. It does not support auto-negotiation to any other speed, and runs in full duplex mode only. The OSA-Express 10 GbE LR feature is defined as CHPID type OSD.

**OSA-Express3 10 GbE LR**
This feature occupies one slot in an I/O cage and has two ports that connect to a 10 Gbps Ethernet LAN through a 9-micron single mode fiber optic cable terminated with an LC Duplex connector. Each port on the card has a PCHID assigned. The feature supports an unrepeated maximum distance of 10 km.

Compared to the OSA-Express2 10 GbE LR feature, the OSA-Express3 10 GbE LR feature has double port density (two ports per feature) and improved performance for standard and jumbo frames. It does not support auto-negotiation to any other speed, and runs in full duplex mode only. The OSA-Express3 10 GbE LR feature is defined as CHPID type OSD, which is supported by z/OS, z/VM, z/VSE, TPF, and Linux on System z.
6.27 QDIO architecture

Queued direct I/O (QDIO)
Queued Direct I/O (QDIO) is a highly efficient data transfer mechanism that satisfies the increasing volume of TCP/IP applications and increasing bandwidth demands. It dramatically reduces system overhead, and improves throughput by using system memory queues and a signaling protocol to directly exchange data between the OSA-Express microprocessor and TCP/IP software. SNA support is provided through the use of TN3270 or Enterprise Extender.

QDIO is supported with OSA-Express GbE SX and LX, OSA-Express 1000BASE-T Ethernet, OSA-Express FENET, OSA-Express Token Ring, and OSA-Express 155 ATM MM and SM (when configured for Ethernet or Token Ring LAN emulation).

In QDIO mode, the OSA-Express microprocessor communicates directly with the server's communications program, using data queues in main memory and utilizing Direct Memory Access (DMA).

QDIO components
The components that make up QDIO are Direct Memory Access (DMA), Priority Queuing (z/OS only), dynamic OSA Address Table building, LP-to-LP communication, and Internet Protocol (IP) Assist functions.
**QDIO versus non-QDIO**

Figure 6-27 illustrates the much shorter I/O process of the QDIO-enabled feature compared with non-QDIO (which has the same I/O path as the OSA-2 features). Consequently, I/O interrupts and I/O path-lengths are minimized. The advantages of using QDIO are: 20% improved performance versus non-QDIO; the reduction of System Assist Processor (SAP) utilization; improved response time; and server cycle reduction.

**QDIO exploiters**
The QDIO exploiters are:

**HiperSockets**  
The HiperSockets implementation is based on the OSA-Express Queued Direct Input/Output (QDIO) protocol, hence HiperSockets is called internal QDIO (iQDIO). The LIC emulates the link control layer of an OSA-Express QDIO interface. HiperSockets copies data synchronously from the output queue of the sending TCP/IP device to the input queue of the receiving TCP/IP device by using the memory bus to copy the data, via an I/O instruction.

**OSA-Express**  
This integration of channel path with network port makes the OSA-Express a unique channel, recognized by the hardware I/O configuration as one of the following channel types:

- OSD (Queued Direct I/O)
- OSE (non-Queued Direct I/O)

**FCP mode**  
FICON channels in FCP mode use the Queued Direct Input/Output (QDIO) architecture for communication with the operating system. The QDIO architecture for FCP channels derives from the QDIO architecture that had been defined for the OSA-Express features and for HiperSockets communications. It defines data devices that represent QDIO queue pairs, consisting of a request queue and a response queue. Each queue pair represents a communication path between an operating system and the FCP channel. It enables an operating system to send FCP requests to the FCP channel via the request queue. The FCP channel uses the response queue to pass completion indications and unsolicited status indications to the operating system.
6.28 HiperSockets connectivity

z9 EC, z9 BC, z990, and z890 servers support up to 16 HiperSockets being defined. Enabling HiperSockets requires the definition of a CHPID as type=IQD using HCD and IOCP. This CHPID is treated like any other CHPID, and is counted as one of the available channels within the z9 EC, z9 BC, z990, and z890 servers.

The HiperSockets LIC on z9 EC, z9 BC, z990, and z890 servers supports:

- Up to 16 independent HiperSockets
- For z/OS, z/VM, Linux, and VSE/ESA, the maximum number of TCP/IP stacks or HiperSockets communication queues that can concurrently connect on a single z9 EC, z9 BC, z990, or z890 server is 4096.
- Up to 12288 I/O devices across all 16 HiperSockets
- Up to 16000 IP addresses across all 16 HiperSockets

A total of 16000 IP addresses can be kept for the 16 possible IP address lookup tables. These IP addresses include the HiperSockets interface, as well as Virtual IP addresses (VIPA) and dynamic Virtual IP Addresses (DVIPA) that are defined to the TCP/IP stack.

Each HiperSocket uses a Channel Path ID (CHPID). The values 03, 04, and 05 (shown in Figure 6-28) are the CHPIDs assigned to the HiperSockets.

Figure 6-28  Spanned and non-spanned HiperSockets defined
With the introduction of the new channel subsystem, transparent sharing of HiperSockets is possible with the extension to the Multiple Image facility (MIF). HiperSockets channels can be configured to multiple channel subsystems (CSS). They are transparently shared by any or all of the configured logical partitions without regard to the CSS to which the partition is configured. Figure 6-28 on page 425 reflects spanned HiperSockets defined on a z9 EC, z9 BC, z990, or z890 server.

zSeries HiperSockets is a technology that provides high-speed TCP/IP connectivity between virtual machines (under z/VM) within System z servers. It eliminates the need for any physical cabling or external networking connection between these virtual machines.

zSeries HiperSockets is Licensed Internal Code (LIC) of the zSeries server, and the communication is through system memory. The connections between the virtual machines form a virtual LAN. HiperSockets uses internal Queued Direct Input/Output (iQDIO) at memory speed to pass traffic between the virtual machines.

HiperSockets can be used to communicate among consolidated images in a single System z server. All the hardware boxes running separate Linux servers can be eliminated and a much higher level of network availability, security, simplicity, manageability, performance, and cost effectiveness is achieved, as compared with servers communicating across a LAN.
6.29 Hardware Configuration Definition (HCD)

HCD processing
To define the I/O configuration for the channel subsystem, run the Input/Output Configuration Program (IOCP). To run IOCP, you need to specify:

- Channel subsystem
- Logical partitions
- Channel paths on the server complex
- Control units attached to the channel paths
- I/O devices assigned to the control units

The Hardware Configuration Definition (HCD) can help you in such tasks; refer to “What is HCD” on page 492 for more information.
Virtualization and Logical Partition (LPAR) concepts

This chapter explains virtualization concepts in general and in detail one type of virtualization in the z196 CPC, that is, Logical Partitioning (LPAR).

Along these explanations, other IBM virtualization implementations are lightly described, such as z/VM, Power/VM and VTS.

In the LPAR part, we cover the concept of a logical partition (LP), and how to define it in a z196 processor. Here we use the acronym LPAR to describe the LIC software that partitions the CPC, creating the virtualized logical partitions.

LPAR allows an operator to allocate CPC hardware resources (including PUs, central storage, and channel paths) among logical partitions.

Then, in LPAR mode, the resources of a server complex can be distributed among multiple control programs (operating systems) that can run on the same server complex simultaneously. Each control program has the use of resources defined to the logical partition in which it runs.
7.1 Virtualization definitions

Virtualization is the process of presenting computing resources in ways that users and applications can easily get value out of them, rather than presenting them in a way dictated by their implementation, geographic location, or physical packaging.

Virtualization is the ability for a computer system to share resources so that one physical server can act as many virtual servers.

Virtualization definitions

Virtualization is a critical component of IT virtualization. Benefits of server virtualization—such as improved utilization, flexibility, responsiveness, workload mobility and disaster recovery—can be limited if only part of the infrastructure is virtualized. Storage virtualization is also one of the foundations for effective cloud deployment. IBM offers a variety of storage virtualization solutions to meet the needs of companies of all sizes.

Virtualization is the ability for a computer system to share resources so that one physical server can act as many virtual servers. Creating many virtual machines consisting of virtualized processors, communications, storage, and I/O devices can reduce administration costs and the overhead of planning, purchasing, and installing new hardware to support new workloads. Through the “magic” of virtualization, software running within the virtual machine is unaware that the “hardware” layer has been virtualized. It believes it is running on its own hardware separate from any other operating system. This can reduce the number of processors and hardware devices needed.

It is important to note (for completeness of the definition) that splitting a single physical entity into multiple virtual entities is not the only method of virtualization. For example, combining multiple physical entities to act as a single, larger entity is also a form of virtualization, and grid computing is an example of this kind of virtualization. The grid virtualizes heterogeneous and geographically dispersed resources, thus presenting a simpler view.
7.2 Virtualization concepts

Virtualization concepts

Virtualization is a very old technique (IBM mainframe have had such since the late sixties) that creates virtual resources and “maps” them to real resources. In other words, separates presentation of resources (virtual resources) to users from actual physical resources.

It aggregates pools of resources for allocation to users as virtual resources showing the way back to a centralized topology. It is a logical solution for collapsing many physical servers into less and more powerful ones.

Summarizing, there are many benefits gained from virtualization, as follows:

- Better hardware utilization (85% of server capacity average daily is not used in a distributed server farm environment). Without virtualization a resource available in one server cannot be migrated to another server, where such resource is lacking.
- Lower power consumption (50% of total data center energy is spent on air conditioning in a server farm) and dynamic energy optimization use.
- Reduced IT management costs. Virtualization can improve staff productivity by reducing the number of physical resources that must be managed by hiding some of the resource complexity, simplifying common management tasks through automation, better information and centralization, and enabling workload management automation.
- Consolidation of application and operating system copies. Virtualization enables common tools to be used across multiple platforms and also enables multiple applications and operating systems to be supported in one physical system. Virtualization consolidates...
servers into virtual machines on either a scale-up or scale-out architecture. It also enables systems to treat computing resources as a uniform pool that can be allocated to virtual machines in a controlled manner.

- Better SW investment protection.
- Simplified Continuous Availability/Disaster Recovery solutions. It also enables physical resources to be removed, upgraded, or changed without affecting users.
- Improved security by enabling separation and compartmentalization that is not available with simpler sharing mechanisms, and that provides controlled, secure access to data and devices. Each virtual machine can be completely isolated from the host machine and other virtual machines. If one virtual machine crashes, none of the others are affected. Virtualization prevents data from leaking across virtual machines, and ensures that applications communicate only over configured network connections.

On the other hand, there are a few problems with virtualization, such as:

- Increased complexity that may endanger Continuous Availability and problem determination
- More overhead when using resources
7.3 Virtualized physical resources

Primarily virtualization logic is accomplished by software (hypervisors such as z/VM, PowerVM) and/or LIC (the IBM VTS tape virtualizer). LPAR is an example of an hypervisor that is an LIC.

In a sense, virtual resources are proxies for real resources (as the ones pictured in Figure 7-3) with the same architecture interfaces and functions, but different attributes.

The resources that may be virtualized are:

- Central storage. When central storage was not virtualized and, for example, we have available 20 MB for application programs, then only 20 programs of 1 MB each, can fit concurrently in central storage. In this case, the multiprogramming level of concurrent business transactions executing these programs is small (about 20). The effect of such a small multiprogramming level is low processor utilization and a smaller number of executed I/Os.

On top of that, in a commercial environment usually only 20% of a program is executed frequently, which indicates waste when using central storage resource. By virtualizing central storage, the programs have the illusion of having much more addresses in central storage than bytes in reality. Then, more programs can fit together, causing a higher multiprogramming level, and more processor utilization and more I/O operations.

It is important to note that central storage virtualization has an operating system scope, that is, the virtualization is done among the programs running within such an operating system.
system copy. Among different operating systems in the same CEC, central storage is sliced among them, but with the possibility of reconfiguring manually from one image to the other.

- Processors. They are virtualized by hypervisors running underneath operating system images in the same CPC. Refer to “IBM hypervisors” on page 439. An example can be LPAR (an hypervisor) delivering physical processors to a z/OS running in an image under such an LPAR. In other words, the physical processors are shared between copies of operating system images. If we have a peak on demand for processors in one image, automatically the hypervisor delivers the majority of physical processors to that image. If the peak moves away to another image, the same behavior is repeated. However, if the adding of all demands is greater than the total physical processor capacity, then the installation must deliver a criteria (weights) indicating the relative importance of each image. It is also possible to dedicate some processors to certain images.

- I/O channels. They can be shared (a sort of virtualization) automatically among images of the same CPC under an LPAR for z/VM. The same FICON channel may have concurrent I/Os coming from different images depending on the I/O demand. The implementation of this sharing is done at the channel level, not at the operating system level. Here, the major purpose is to keep the recommended level of connectivity between a z/OS and an I/O device, but with less physical I/O channels. Sometimes virtual devices are not backed by physical devices, for example, along channel virtualization there is a total emulation of the hardware, as in the case of HypeSockets and IC Coupling Facility links.

- Disk drivers. Hypervisors, such as z/VM, may slice a DASD volume into virtualized minidisks. It is the same approach as slicing central storage among images in the same CPC.

- Tape drivers. There are implementations, such as IBM VTS, where tape drivers are virtualized in DASD. The application programs have the illusion of writing to a tape, but the I/O requested is redirected to a real DASD device, which works as a cache of real tape devices. The rationale is to avoid the mechanical delays of a tape volume, that is, rewind, load, search for a file sequentially.

- Networks. Virtualization can be viewed as a collection of capabilities that virtualize the network resources, such as IP addresses, network adapters, LANs, bandwidth management, and more. By virtualizing networks, customers can pool and share network elements to make communication across their IT infrastructure faster, more efficient, cost effective, secure, and available. Virtualized networks are also more flexible, allowing customers to implement changes to the infrastructure, as required, so they can adapt to business needs in real time, with minimal downtime. Virtual LAN (VLAN) technology uses software to configure a network of computers to behave as if they were physically connected, even though they are not.
7.4 Hypervisor types

Hypervisor types

An hypervisor is a sort of operating system. Its major role is to virtualize physical resources such as processors, central storage and some I/O devices to other operating systems running underneath. There are two types of hypervisors:

- **Type-1 hypervisors** have the function implemented as part of the operating system software or the hardware firmware on a given hardware platform. When the hypervisor is implemented as part of the operating system, this operating system providing the hypervisor function is called the “host” operating system. The operating system running within a virtual machine (VM) created on the host system is called the “guest” operating system. Figure 7-4 illustrates a Type-1 hypervisor. The hypervisor box shown in the center of the figure is either the firmware hypervisor (physical hypervisor) or an operating system (host operating system) with the hypervisor function built in. The boxes labelled OS are the virtual machines (guest OS). z/VM and LPAR are examples of a Type-1 hypervisor. Some other examples are the XEN OpenSource Hypervisor, VMWare ESX Server, Virtual Iron, and ScaleMP.

- **Type-2 hypervisors** are the ones running on a host operating system as an application. In this case, the host operating system refers to the operating system on which the hypervisor application is running. The host operating system runs directly on the hardware. The operating system running inside the virtual machine (VM) created using the hypervisor application is the guest operating system in this context. Figure 7-4 illustrates a Type-2 hypervisor. The box labelled Host OS is the host operating system. The box labelled hypervisor is the hypervisor function running as an application on the host.
operating system. The other OS boxes are the virtual machines (guest operating systems) running on top of the hypervisor application running on the host operating system. VMWare GSX, Microsoft Virtual Server, Win4Lin, and UserModeLinux are some examples of this type of hypervisor.
7.5 Hypervisor technologies (I)

Hypervisor technologies
It is useful to discuss the hypervisor technologies currently used.

Trap and emulate method
In this method, a guest operating system runs in user mode (also called problem mode in z terminology) and the hypervisor runs in privileged mode. Privileged instructions issued by guest operating systems are trapped by the hypervisor. This technology was originally used by mainframes in the 1960s and 1970s (VM/370). Figure 7-5 illustrates this method. The advantage of this method is that the guest operating system runs unmodified. The problem is the substantial overhead generated. Examples are CP67 and VM/370.

The translate, trap and emulate method
This method is almost identical to the trap and emulate method. The difference is that some of the guest instructions must be replaced with trap operations, so some guest kernel binary translation may be required. The benefit is that the guest operating system runs unmodified. The problem is the substantial overhead generated. The translate, trap and emulate method is illustrated in Figure 7-5. Examples are VMWare and Microsoft VS.
7.6 Hypervisor technologies (II)

Hypervisor call method (paravirtualization)

In this method (also known as paravirtualization), a guest operating system runs in privileged mode and the hypervisor runs in super-privileged mode. This method is illustrated in Figure 7-6. The guest operating system kernel (for example, AIX, i5/OS®, or Linux) is modified to do hypervisor calls for I/O, memory management, yield rest of time slice, and so on. Memory mapping architecture is used to isolate guests from each other and to protect the hypervisor. The benefit is high efficiency and the problem is to modify the guest operating system for calling the hypervisor. Examples are XEN and the PowerVM hypervisor.

Direct hardware support method

In this method, the guest operating system runs in privileged mode and the hardware runs most of the virtualization. The guest operating system can be run unmodified, but can issue some hypervisor calls to improve performance or capability such as I/O (z/VM), yield time slice, and get HiperDispatch information (PR/SM and z/VM). This method provides extensive hardware assists for hypervisor (virtual processor dispatching, I/O pass-through, memory partitioning, and so on).

The direct hardware support method, illustrated in Figure 7-6, is used by the IBM System z and zSeries (PR/SM and z/VM) family of mainframes.
7.7 IBM hypervisors

As pictured in Figure 7-7, the IBM hypervisors are LPAR (contained in PR/SM), z/VM, Power/VM, and a Statement of Direction (SOD) for KVM (an hypervisor running in x86 Blades). Starting with z10 it is not possible to have a z CPC without LPAR being active. Then, z/VM must run under LPAR, as shown in the picture.

LPAR is an LIC (hidden software) hypervisor extensively covered in this chapter starting at “CPC in LPAR mode” on page 450.
7.8 z/Virtual Machine (z/VM)

z/VM is a chargeable hypervisor allowing the sharing of the mainframe’s physical resources such as processors, memory, network adapters and disks. All these resources are managed by z/VM.

z/VM provides each user with an individual working environment known as a virtual machine. The virtual machine simulates the existence of a dedicated real machine, including processor functions, memory, networking, and input/output (I/O) resources. Operating systems and application programs can run in virtual machines, as guests. For example, you can run multiple Linux and z/OS images on the same z/VM system that is also supporting various applications and users. As a result, development, testing, and production environments can share a single physical computer.

A virtual machine uses real hardware resources, but even with dedicated devices (like a tape drive), the virtual address of the tape drive may or may not be the same as the real address of the tape drive. Therefore, a virtual machine only knows “virtual hardware” that may or may not exist in the real world. Figure 7-8 shows the layout of the general z/VM environment.

z/VM’s hypervisor is called Control Program (CP). CP is a software hypervisor. When the user logs into z/VM, the CP creates a virtual machine which runs an operating system called Conversational Monitor System (CMS). This operating system can run only under the CP.
Through CMS:

- z/VM system programmers can manage and control z/VM and its virtual machines.
- Programmers can develop application source code.

CP can run an unlimited number of images called virtual machines executing z operating systems such as z/OS, z/TPF, Linux, z/VSE, CMS, or z/VM (forming a cascade).

CP (as LPAR) uses the Start Interpretive Instruction (SIE) to associate the physical PU with the Virtual PU. The virtual PU is the PU as perceived by the operating system running in the virtual machine.

The major reason currently to have z/VM is to allow an unlimited number of zLinux virtual machines and to enjoy the user-friendly interfaces of CMS.
7.9 z/VM options in HMC

z/VM options in HMC
Allows basic z/VM functions to be performed from HMC. Starting with z/VM 5.3, no network connection is required.

This function uses the SCLP hardware interface to access system management z/VM APIs. Some of the supported operations are:

- View z/VM guests
- Activate or deactivate z/VM guests
- Display guest status configuration
7.10 Virtualization in zBX blades

Virtualization in zBX

In the zBX blades there are two types of hypervisors: PowerVM (also called pHyp) used in IBM Power RISC blades, and KVM (also called xHyp) used in the SOD x blades. Some properties of these are:

- **pHyp**, the hypervisor for AIX virtual servers:
  - Scales to 8 cores per blade
  - Larger number of fast processing threads
  - Floating point accelerators

- **xHyp**, the hypervisor for Linux virtual servers (a statement of direction):
  - Scales to 8-12 cores per blade
  - Fast processing threads
  - Commodity I/O
  - Modest qualities of service

In this chapter we just cover PowerVM. Refer to “PowerVM virtual servers” on page 444,
7.11 PowerVM virtual servers

PowerVM virtual servers
The PowerVM hypervisor manages the processors, memory, network, disk, and ISO images that are provided to virtual servers (also called partition). Those virtual servers are the computing elements that customers use to deploy their applications.

Virtual servers on an IBM Power Blade are defined in terms of processors, memory, and I/O resources they are allowed to consume.

Virtual server processor resource
A virtual server may have either shared or dedicated virtual processors. For the ones with shared processors, there is a fine-grained allocation mechanism.

To manage the distribution of shared processor resources among virtual servers, you may use several controls:

- Entitlement capacity (looks like the LPAR weight concept). Through a time slice mechanism PowerVM hypervisor associates physical processors with virtual processors based on each virtual server entitlement capacity. The guarantee is defined by the installation through a fraction of the shared physical processors. Each of the eight blade processors has the capacity of 1.0 processing unit (processor). A shared virtual processor is allocated between 0.1 and 1.0 PUs. Each dedicated processor eliminates the capacity for 10 shared virtual processors.
In Figure 7-11 on page 444, we see that the virtual server running AIX 5.3 has an entitlement capacity guarantee of 3.8 processors, in a total of six shared physical processors in the blade.

- Number of virtual processors. Another way of delimiting the amount of processor resource delivered to a virtual server is by defining a number of virtual processors less than the number of shared physical processors. Figure 7-11 on page 444 shows that the virtual server running AIX 5.3 has four virtual processors, in a total of six shared physical processors in the blade.

- Capping. This means limiting the consumed processor capacity to the values defined at entitlement capacity.

- Partition weight (is not the same concept as LPAR weight). All uncapped partitions share the unused capacity based on user-defined weighting. Weight scale is 0 to 255. Figure 7-11 on page 444 shows that the virtual server running AIX 5.3 has a weight equal to 50 in a total weight of 70.

**Virtual server memory**
The virtual server memory is dedicated. The minimum amount is 128 MB to the value of the total central storage available capacity.

**Virtual server devices**
Virtual SCSI devices are supported and associated with dedicated LUNs. A virtual server requires a boot device, which can be a virtual disk, a virtual optical device, or a virtual network adapter.
7.12 Comparing hypervisor terminology

- LPAR: Logical Partition, Logical CPU, LPAR Hypervisor, Weight
- z/VM: Virtual Machine, Virtual CPU, Control Program, Weight
- PowerVM: Virtual server, Virtual Processor, PowerVM Hypervisor, Entitlement Capacity

Figure 7-12  Hypervisor terminology

**Hypervisor terminology**

Figure 7-12 shows the different terms used in three IBM hypervisors, as follows:

- The first one is the name of the partition created by the hypervisor.
- The second one is the name of the virtualized shared processor.
- The third one is the name of the hypervisor itself.
- Finally, the name of the mechanism used by the installation to control the amount of processor capacity delivered to each partition.
7.13 History of operating environments

- Uniprocessors (UP)
- Multiprocessors (MP) and physical partitioning
- Introduction of PR/SM and LPAR mode
- Base sysplex
- Parallel Sysplex
- Intelligent Resource Director

Figure 7-13  History of operating environments

History of operating environments
Here, we are changing gear. In preparation for covering LPAR, we describe historically the different design implementation of mainframes.

Uniprocessors
When S/360 was first announced, the available hardware at the time was a single CP server containing less storage and fewer MIPS than the cheapest pocket calculator available today—and it was also considerably larger and more expensive! As the business world discovered more and more uses for this new tool, the demand for MIPS outpaced the rate at which CP speed was progressing.

Multiprocessors
As a result, IBM introduced the ability to add a second CP to the server. This provided more power, and potentially more availability, since you could conceptually continue processing even if one of your CPs failed. These machines were available either as Attached Processors (APs), where only one CP had an I/O subsystem, and multiprocessors (MPs), where each CP had access to its own I/O subsystem.

In addition to providing more capacity on an MP, the servers, I/O channels, and storage could be physically “partitioned”, meaning that two separate copies of the operating system could be run on the servers, if desired.
The next major hardware advance, in terms of flexibility for running multiple copies of the operating system, was Processor Resource/System Manager (PR/SM) with the LPAR feature, introduced on the IBM 3090 range of processors. PR/SM provided the ability, even on a server with just one CP, to run up to four logical partitions (LPs). This meant that you could split your production applications across several system images, and have a separate development system, or a test system, all on a server with a single CP. Such a configuration did not provide much protection from CP failures (if you had just one CP), but it did help protect against software failures. It also provided the ability to create a test environment at a lower cost, thus enabling you to ensure that all software was tested before your production applications were run on it.

**Base sysplex**

All the enhancements to this point were aimed at providing the ability to break a single, large system into a number of smaller, independent system images. However, as applications grew and system management became a concern, a mechanism was required to provide closer communication with, and control of, the systems. To address this need, MVS/ESA Version 4.1 introduced the concept of a “sysplex” (now called a Base sysplex, to differentiate it from a Parallel Sysplex). This provided a new MVS component known as the Cross System Coupling Facility (XCF), which allows applications running on multiple images to work more cooperatively without having to incur significant overhead or complex programming. This step laid the foundation for data sharing, which was introduced by the next significant advance called Parallel Sysplex, introduced with MVS/ESA Version 5.1.

**Parallel Sysplex**

Parallel Sysplex allows a single image infrastructure to have multisystem data sharing with integrity, availability, and scalability not possible with earlier data sharing mechanisms. These benefits are enabled by a new external operating system known as a Coupling Facility Control Code (CFCC). Coupling Facilities were initially run on dedicated servers (9674s), but have since been enhanced to run in special LPs on the general purpose 9672s and, more recently, zSeries and z9/z10 and z196 CPCs running in specialty engines called IFL.

**Intelligent Resource Director (IRD)**

This brings us to the present, with the announcement of the zSeries and z9 servers and the z/OS operating system, and Intelligent Resource Director. A typical medium-to-large S/390 configuration contains a variety of server types and sizes, generally operating in LPAR mode, and supporting images that run batch, OLTP, web servers, application development, Enterprise Resource Planning (such as SAP R/3), Business Intelligence, and various other workloads. Within each LP, WLM in Goal mode is responsible for allocating resources such that it helps the most important workloads (as specified by the installation) meet their Service Level Agreement objectives. WLM has been providing this capability since MVS/ESA 5.1, and is generally considered to be very effective at this task. Moving up a level, there is PR/SM Licensed Internal Code (LIC), with responsibility for allocating the physical processor resource, based upon an installation-specified set of weights for the LPs.

So, WLM manages the allocations of the resources that are given to the LP by PR/SM, and PR/SM divides processing resources among the LPs on the server. WLM knows the relative importance of the work running in each LP, and is ideally placed to decide what the weight of each LP should be, so that PR/SM will give the CPU to whichever LP needs it the most in order to help meet the goals of the business.

Allowing WLM and PR/SM to communicate is one of the functions delivered by IRD.
7.14 CPC in basic mode

CPC in basic mode
When a server is in basic mode, all server resources (CPs, storage, and channels) are available to the one operating system. All the physical CPs are used in dedicated mode for the one operating system. Any excess CP resource is wasted, since no other system has access to it.

Situations still exist where servers are run in Basic mode (for example, if the z/OS system needs to use the entire capacity of the server). However, because of the huge capacity of modern servers, this is becoming less and less common.

Since z10 (and z196), the Basic mode is not allowed anymore.
7.15 CPC in LPAR mode

Processor Resource/System Manager (PR/SM)
PR/SM is a standard feature of z196 CPC. It is formed by:

- Multi High Performance Guest Support (MHPGS) in VM
  This allows several preferred guests under z/VM, all with performance very close to that available when running native.
- Logical Partitioning (LPAR)
  It is a software LIC located in HSA. It allows a CPC to be divided into multiple logical partitions (LP), that is, virtualization. This capability was designed to help isolating workloads in different z/OS images, so you can run production work separately from test work, or even consolidate multiple CPCs into a single CPC.

An LP has the following properties:

- Each LP is a set of physical resources (processor, storage, and channels) controlled by just one independent image of an operating system, such as z/OS, Linux, CFCC, z/VM, z/TPF, or VSE.
- You can have up to 60 LPs in a z196 CPC.
- Each LP is defined through IOCP/HCD. For example, the IOCP RESOURCE PARTITION = ((LP1,1),(LP2,2)) statement defines two LPs. A power-on reset (POR) operation is required to add or remove LPs. However, now through reserved LPs, it is possible to add an LP without the need of a POR.
- LP options, such as the number of logical CPs, the LP weight, whether LPAR capping is to be used for this LP, the LP storage size (and division between central storage), security options, and other LP characteristics are defined in the Activation Profiles in the HMC.
- Individual physical CPs can be shared between multiple LPs, or they can be dedicated for use by a single LP.
- Channels can be dedicated, reconfigurable (dedicated to one LP, but able to be switched manually between LPs), or shared.
- The CPC storage used by an LP is dedicated, but can be reconfigured from one LP to another with prior planning and without stopping the both logical partitions.
- Although it is not strictly accurate, most people use the terms LPAR and PR/SM interchangeably to refer to the software located in HSA. Similarly, many people use the term LPAR when referring to an individual logical partition. However, the acronym LP is technically more accurate for the logical partition image. LPAR should be used for the LIC code.
7.16 Shared and dedicated logical CPs example

Logical and physical CPs

Physical CPs can be dedicated or shared, as shown by the 10 CPs in Figure 7-16. If dedicated, the physical CP is permanently assigned to a logical CP of just one LP. The advantage of this is less LPAR overhead. An operating system running on a server in an hypothetical Basic mode gets marginally better performance than the same server running z/OS as a single LP with dedicated CPs. This is because, even with dedicated CPs, LPAR still gets called whenever the LP performs certain operations (such as setting the TOD clock).

If you share physical CPs between LPs rather than dedicating them to a single LP, there is more LPAR overhead. The LPAR overhead increases in line with the proportion of logical CPs defined in all the active LPs to the number of shared physical CPs.

IBM has a tool (LPARCE) which estimates the overall LPAR overhead for various configurations. Your IBM marketing representative can work with you to identify the projected overhead of various configurations and workload mixes. If you are already in LPAR mode, RMF reports this overhead in the LPAR Activity report.

While the use of shared CPs does cause more overhead, this overhead is nearly always more than offset by the ability to have one LP utilize CP capacity that is not required by another sharing LP. Normally, when an operating system that is using a shared CP goes into a wait, it releases the physical CPs, which can then be used by another LP. There are a number of controls available that let you control the distribution of shared CPs between LPs.

It is not possible to have a single LP use both shared and dedicated CPs.
One of the significant reasons for the increased number of LPs is server consolidation, where different workloads spread across many small machines may be consolidated in LPs of a larger server. LPAR also continues to be used to run different environments, such as system programmer test, development, quality assurance, and production, in the same server.

**LPAR CP management**

LPAR management of the shared CP environment is described here in some detail.

Figure 7-16 on page 452 shows a 10-CP IBM z196 16-way CPC. We use the term "physical CP" to refer to the actual CPs that exist on the CPC. We use the term "logical CP" to refer to the CPs each operating system perceives in order to dispatch work (TCB/SRB). The number of logical CPs in an LP must be less than or equal to the number of physical CPs.

As shown in Figure 7-16 on page 452, two CPs are dedicated to LP MVS1. The two dedicated CPs are for use exclusively by MVS1. For this LP then, the number of physical CPs is equal to the number of logical CPs.

The remaining eight physical CPs are shared between the LPs: MVS2, MVS3, MVS4, and MVS5. Each of these LPs can use any of the shared physical CPs, with a maximum at any one time equal to the number of online logical CPs in that LP. The number of logical CPs per LP is:

- Six logical CPs in LP MVS2
- Three logical CPs in LP MVS3
- Two logical CPs in LP MVS4
- Two logical CPs in LP MVS5

The number of shared physical CPs can be less than the sum of logical shared CPs in all the LPs. On the other hand, an MVS operator could vary logical CPs online and offline, as if they were physical CPs. This can be done through the z/OS CONFIG command.

An LP cannot have more logical CPs online than the number defined for the LPs in the HMC.

**Note:** Due to the implementation of Capacity Upgrade on Demand (CUoD), there is the ability to define reserved logical CPs for an LP. These reserved logical CPs may be associated with future physical CPs that may be installed nondisruptively on the CPC, and can then be nondisruptively exploited by the LP.

In order to determine the correct number of logical CPs in an LP, take the following variables into consideration: LPAR overhead; the level of parallelism you want; correct use of the assigned weight; the number of physical CPs used by the LP; the desired logical CP speed. Sometimes the correct value in the night shift is not the best for the mid-morning workload, and at other times, it varies from one second to another. Refer to z/OS Intelligent Resource Director, SG24-5952, for more comprehensive information about this topic. Another recommendation is to implement the HiperDispatch functionality, that as a by-product determines the right number of online logical CPs.

All the ideas described here apply also to zAAP and zIIP, not only to CP in one MVS LP.
7.17 LPAR dispatching and shared CPs

LPAR logical CP dispatching

The code that provides the LPAR dispatching function (associates a physical processor with a logical CP) is called LPAR Scheduler Licensed Internal Code (LIC). LPAR LIC logic executes on all of the physical CPs. LPAR LIC dispatches a logical CP on a physical CP (the one the LPAR LIC is currently running on) by issuing the Start Interpretive Execution (SIE) instruction, with the logical CP represented by a state description control block (SDCB), as a parameter. There is one SDCB per each logical CP.

This causes the operating system code or application code in the LP to execute on the physical CP, through the logical CP. The logical CP is dispatched on the physical CP by copying the LP's logical CP status (PSW, registers, and so forth) from the specific SDCB to the corresponding actual entities.

When the logical CP is intercepted (the association of physical CP with the logical CP ends temporarily), the logical CP status is saved in SDCB and the LPAR LIC is automatically dispatched on the physical CP again. This code then chooses another logical CP to dispatch, and the whole process starts again.

Conceptually, this process is similar to the way that the z/OS dispatcher works. For example:

- The LPAR Scheduler LIC corresponds to z/OS dispatcher code.
- Each logical CP corresponds to an z/OS dispatchable unit (task, or service request).
- The logical CP state information stored in the SDCB corresponds to the TCB or SRB.
- The SIE instruction corresponds to the LPSW (Load PSW) instruction.
An intercept of a logical CP corresponds to an interrupt for a z/OS dispatchable unit. Information about the possible motifs for intercepts is provided in 7.18, “Reasons for intercepts” on page 456.

Figure 7-17 on page 454 illustrates the flow of execution, where we have eight shared physical CPs. Let us examine this now.

**Logical CP dispatching**

Every logical CP represents a dispatchable unit to LPAR LIC. In Figure 7-17 on page 454, MVS2 has 6 logical CPs, MVS3 has 3 logical CPs, MVS4 has 2 logical CPs, and MVS5 also has 2 logical CPs. This gives us a total of 13 logical CPs, so LPAR LIC has up to 13 dispatchable units to manage in our environment. LPAR LIC is responsible for dispatching logical CPs on physical CPs.

When a logical CP is ready to run (not in wait), it is placed on the logical CP ready queue. This queue is ordered by a priority that is based on the LP weight and the number of logical CPs, as declared by the installation. This is discussed in more detail in z/OS Intelligent Resource Director, SG24-5952.

**Logical CP execution**

So how does a logical CP move from being on the ready queue to executing on a physical CP, and where is the processing performed that does this? As mentioned before, LPAR LIC, sometimes referred to as the LPAR scheduler, is responsible for dispatching a logical CP on a physical CP. LPAR LIC executes on each physical CP. When it is ready to dispatch a logical CP on a physical CP, LPAR LIC issues a SIE instruction, which switches the LPAR LIC code from the physical CP and replaces it with the code running on the logical CP.

The steps that occur in dispatching a logical CP, illustrated in Figure 7-17 on page 454, are as follows:

1. The next logical CP to be dispatched is chosen from the logical CP ready queue based on the logical CP weight.
2. LPAR LIC dispatches the selected logical CP (LCP5 of MVS2 LP) on a physical CP in the server (CP0, in the visual).
3. The z/OS dispatchable unit running on that logical CP (LCP5 of MVS2) begins to execute on physical CP0. It executes until its time slice (generally between 12.5 and 25 milliseconds) expires, or it enters a voluntary wait (lack of service in MVS2), or it is intercepted for some reason.
4. As shown, the logical CP keeps running (for example) until it uses all its time slice. At this point the logical CP environment is saved in SDCB and control is passed back to LPAR LIC, which starts executing on physical CP0 again.
5. LPAR LIC determines why the logical CP ended execution and requeues the logical CP accordingly. If it is ready with work, it is requeued on the logical CP ready queue and step 1 begins again.

This process occurs on each physical CP. As shown in Figure 7-17 on page 454 with 8 physical CPs shared, LPAR LIC code executes in each of them. This explains how a logical CP is dispatched from the logical CP ready queue to a physical CP.
7.18 Reasons for intercepts

- A logical CP continues processing on a physical CP until one of the following events (intercepts) occurs:
  - Its time slice ends (12.5 - 25ms)
  - It enters a processor wait state
  - When it runs over its weight target, it is preemptable by an I/O for an underweight target logical CP
  - z/OS starts to spin waiting for an event (for example, waiting for locks) - in this case, it will give up its current time slice

- The duration of a time slice is based on either:
  - User option
    - User selects a time slice interval on the HMC
  - LPAR dynamically determined
    - LPAR determines dynamically best value (recommended)

Intercept reasons
Every time that a logical CP is taken away from a physical CP, we have an intercept. The causes for an intercept are:

- End of the time slice. A time slice is the maximum length of time that a logical CP is dispatched on a physical CP. The use of time slicing ensures that a task in a loop cannot cause severe performance problems in other LPs by monopolizing a physical CP.
  
The default duration for a time slice is limited to between 12.5 and 25 ms. The user can override this and specify their own duration for a time slice. However, this is not recommended because it is unlikely that the user's specification is more efficient.
  
The formula used for the default time slice is:
  
  \[
  \text{time slice} = \frac{25 \text{ ms} \times \text{number of physical CPs}}{\text{total number of logical CPs not in stopped state}}
  \]

- When it is running over its weight target, a logical CP is preemptable by an I/O for an underweight target logical CP. See “LPAR weights” on page 462 to get more information about this topic.

- z/OS is starting a spin loop and voluntarily gives up its current time slice. z/OS knows that it is functioning under LPAR. This is so that it can give control of a physical CP back to LPAR LIC in certain circumstances (a spin loop, for example). Generally this happens...
when it is not doing any productive work, and is an example of an event-driven intercept. See “LPAR event-driven dispatching” on page 458, for more information about this topic.

However, in general operations, z/OS behaves as if it is processing on a dedicated server where the number of physical CPs is equal to the number of defined logical CPs. One example of this is the determination of the SRM constant. Even though z/OS knows the number of physical CPs that are actually installed in the CPC, WLM uses an SRM constant (to convert processor seconds to processor service units) that is based on the number of logical CPs in this LP. For consistency this is the best approach; however, it does not take into consideration that the MP effect is related to the number of physical CPs—and not to the number of logical CPs. Then, the SRM constant varies when the operator changes the number of logical CPs online, through the CONFIG command.

- The operating system places the logical CP in a wait (because of a no work-ready situation). This is another case of an event-driven intercept; refer to 7.19, “LPAR event-driven dispatching” on page 458, for more information about the event-driven function of LPAR.

**Note:** The following case is *not* considered an intercept:

If the operating system wants to execute some action not allowed in LPAR mode, such as changing the TOD clock (its contents are global and not local), the LPAR LIC gains control to simulate the action and immediately returns control to the same logical CP.

The simulation provided by LPAR consists of keeping a TOD offset in the descriptor block of the logical CP, so that when z/OS asks for the time, the contents of the real TOD are adjusted by the amount contained in the offset.

**Duration of time slice**

Although we recommend *against* performing these actions, the installation can:

- Set the time slice value in HMC. Rather than letting PR/SM dynamically determine the time slice, you can set a specific value, as low as 1 millisecond. See “LPAR event-driven dispatching” on page 458 for more information about this topic.

- Inhibit event-driven dispatching. See “LPAR event-driven dispatching” on page 458 for more information about this topic.

Again all the ideas covered here apply to CP and the other instance of PUs, such as zAAP, zIIP, IFL, and ICF.
7.19 LPAR event-driven dispatching

**Event-driven dispatching**

*Event-driven dispatching* is the ability of LPAR to take a physical CP away from a logical CP (intercept) before its time slice ends, generally because it is not doing any productive work. All cases of intercepts, except where the time slice ends, are examples of event-driven intercepts. Setting up your LPs to be event-driven (this is the default) is highly recommended because it ensures the most efficient use of CP resources.

The opposite of event-driven is time-driven, where the only time a logical CP will be intercepted is at the end of the time slice. Although this is not recommended, it can still be specified by the installation.

As shown in Figure 7-19, the steps showing event-driven dispatching are as follows:

1. LPAR LIC executing on physical CP0 selects LP MVS2 logical CP5 to dispatch on physical CP0.
2. LPAR LIC dispatches MVS2 logical CP5 on CP0 through the SIE instruction.
3. The z/OS dispatchable unit on MVS2 logical CP5 is executing on physical CP0. If it were to use all its time slice, it would execute for between 12.5 and 25 milliseconds (ms). In this example, it does not use all of its time slice.
4. MVS2 logical CP5 enters a voluntary wait (no ready dispatchable units found in that MVS2) after 8 ms. This is detected by the LPAR LIC and the time slice is ended at 8 ms.
MVS2 logical CP5 is returned to be requeued in a wait queue (waiting for an interrupt) and another ready logical CP is chosen for dispatch on physical CP0. If there is not any ready logical CPU, then LPAR places the physical CP0 in a real wait state.
7.20 LPAR time slice interval

- The user has two options in the HMC to affect the event-driven dispatching function on LPAR, depending the box that is checked:

  - Dynamically determined by the system. This indicates that LPAR is to use defaults
  - Determined by the user
    - In this case, you have to select your own time slice duration for use with event-driven dispatch. This option is not recommended

**Figure 7-20  LPAR time slice Interval**

**Duration of LPAR time slice interval**

The user has two options in the HMC to affect the event-driven dispatching function:

- Time slice value, which can be:
  - Dynamically determined by the system (default). This is selected by checking the box *Dynamically determined by the system*.
    This indicates that the server is to use defaults. That is, event-driven dispatching is turned on and the time slice duration is determined by the system. We highly recommend that you use this default.
  - Determined by the user. This is selected by checking the box *Determined by the user*.
    In this case, you have to select your own time slice duration for use with event-driven dispatching. LPAR still uses event-driven dispatching, but overrides the default time slice duration with the value that the user has specified here.

- Switch off event-driven dispatching by checking the box *Do not end the time slice if a partition enters a wait state*.

If you switch off event-driven dispatching, you must specify a value for the dispatching interval. You can use the normal default values of between 12.5 and 25 milliseconds, or select values of your own. Once again, we do not recommend using this option because there are very few, if any, configurations that will perform better with event-driven dispatching turned off.
The RMF LPAR Activity report shows the status of event-driven dispatching (Wait Completion = NO means that event-driven dispatching is being used), and whether the time slice value is determined dynamically or has been set by the user. This part of the LPAR Activity report is shown here:

<table>
<thead>
<tr>
<th>LPAR ACTIVITY REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAIT COMPLETION</td>
</tr>
<tr>
<td>DISPATCH INTERVAL</td>
</tr>
</tbody>
</table>
7.21 LPAR weights

LPAR weights are used to control the distribution of shared physical CPs between LPs. Therefore, for LPs with dedicated CPs, LPAR weights are not assigned.

LPAR weights determine the guaranteed (minimum) amount of physical CP resource an LP should receive (if needed). This guaranteed figure may also become a maximum when either:

- All the LPs in the CPC are using all of their guaranteed amount (for example, if all LPs were completely CPU-bound).
- The LP is capped using traditional LPAR capping.

An LP may use less than the guarantee if it does not have much work to do. Similarly, it can use more than its weight, if the other LPs are not using their guaranteed amount.

LPAR LIC uses weights and the number of logical CPs to decide the priority of logical CPs in the logical CP ready queue. The following formulas are used by LPAR LIC in the process of controlling the dispatching of logical CPs:

- WEIGHT(LPx)\% = 100 * WEIGHT(LPx) / SUM_of_ACTIVE LPs WEIGHTs
- TARGET(LPx) = WEIGHT(LPx) \% * (# of NONDEDICATE_PHYS_CPs)
- TARGET(LCPx)\% = TARGET(LPx) / (# of LCPs_in_LPx) * 100
**Formula definitions**
The definitions of the formula terms are:

**WEIGHT(LPx)%**
This indicates the percentage of the total shared physical CP capacity that will be guaranteed to this LP. This percentage will vary depending on the weights of all the active LPs. In the example, the MVS2 value is 50%, assuming all the LPs are active.

**TARGET(LPx)**
This indicates, in units of shared physical CPUs, how much CP resource is guaranteed to the LP. This figure cannot be greater than the number of logical CPUs in the LP. This is because you cannot use more physical CPUs than the number of logical CPUs you have defined—each logical CPU can be dispatched on only one physical CPU at a time. So, even if there are eight physical CPUs available, an LP that has been defined with only four logical CPUs can only ever use four of the physical CPUs at one time.

If you specify a weight that guarantees you more capacity than can be delivered by the specified number of logical CPUs, the additional unusable weight will be distributed among the other LPs by LPAR.

In Figure 7-21 on page 462, MVS2’s WEIGHT(LPx)% is 50% of eight physical CPUs, meaning that this LP should be given the guarantee capacity of four physical CPUs.

**TARGET(LCPx)%**
This takes the TARGET(LPx) value (that is, the number of physical CPUs of capacity) and divides that by the number of logical CPUs defined for the LP. The result determines the percentage of a physical CPU that should be given to each logical CPU. This in turn determines the effective speed of each logical CPU. As shown in Figure 7-21 on page 462, the MVS2 value is 4 / 6 * 100, that is, each MVS2 logical CPU will be guaranteed 66% of the capacity of a physical CPU.

Over time, the average utilization of each logical CPU is compared to this value. If Target is less than Current, then the logical CPU is taking more CPU resource than the guarantee and its priority in the ready queue is decreased. It does not mean that it is prohibited from consuming CPU; it simply means that it will tend to sit lower in the queue than other logical CPUs that have used less than their guaranteed share of the CPU resource. Also, these logical CPUs are going to be preemptable by an I/O interrupt for a logical CPU that is behind its target.

If Target is greater than Current, then the logical CPU is taking less CPU resource than the guarantee and its priority in the ready queue is increased. This means that it has a better chance of being dispatched on a physical CPU. Also, these logical CPUs are not going to be preempted by an I/O interrupt for another logical CPU.

The current logical CPU utilization is shown by RMF in the CPU Activity report (LPAR Busy%) and in the LPAR Activity report (Logical Processors Total). This figure includes the processor time used by LPAR LIC within the LP.

**Determine the optimum number of logical CPUs**
Now that we have described how logical CPU executes work (TCB/SRB in a z/OS LP), and we have explained the use of the logical CPU ready queue, we can address the question of what is the optimum number of logical CPUs for an LP.
As shown in Figure 7-21 on page 462, MVS2 has six logical CPs, each of which must get a share of the four physical CPs, as guaranteed by TARGET(LPx). This means that the operating system in MVS2 can run six units of work simultaneously. But it also means that each logical CP does not get a complete physical CP of service (it gets 66%, as we showed previously).

If we change the number of logical CPs that MVS2 has (six) to four, then each logical CP is approximately equal to a physical CP, and will appear to have a higher effective speed. The total amount of CP service the LP receives does not change.

When a logical CP does not equal a physical CP of service, the MIPS per logical CP appears to be less than the MIPS of the physical CP. In Figure 7-21 on page 462, MVS5 is the only LP where a logical CP equals a physical CP. The apparent server speed of each LP is listed in Table 7-1.

<table>
<thead>
<tr>
<th>LPAR name</th>
<th>Weight</th>
<th>Logical CPs</th>
<th>Percent of physical CPs</th>
<th>Physical CPs per LP</th>
<th>MIPS per logical CP</th>
<th>Total MIPS per LP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVS2</td>
<td>50</td>
<td>6</td>
<td>50%</td>
<td>4</td>
<td>133</td>
<td>800</td>
</tr>
<tr>
<td>MVS3</td>
<td>15</td>
<td>3</td>
<td>15%</td>
<td>1.2</td>
<td>80</td>
<td>240</td>
</tr>
<tr>
<td>MVS4</td>
<td>10</td>
<td>2</td>
<td>10%</td>
<td>0.8</td>
<td>80</td>
<td>160</td>
</tr>
<tr>
<td>MVS5</td>
<td>25</td>
<td>2</td>
<td>25%</td>
<td>2</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>Totals</td>
<td>100</td>
<td>13</td>
<td>100%</td>
<td>8</td>
<td>NA</td>
<td>1600</td>
</tr>
</tbody>
</table>

Each logical CP receives a number of time slices on a physical CP. The MIPS that a logical CP delivers depends on how many time slices it receives over a period of time in comparison to the number of time slices available during this time. The number of time slices the logical CP receives depends on the priority in the ready queue, which depends on the LP’s weight and its number of logical CPs.

As shown in the visual, each logical CP in MVS2 gets a smaller number of time slices than each logical CP on MVS5. This means that more concurrent units of work can be dispatched in MVS2 than in MVS5, because MVS2 has six logical CPs. However (assuming that MVS2 and MVS5 are operating at the limit implied by their weights), each logical CP on MVS2 appears to be a slower CP compared to MVS5 because its four physical CPs of service are divided between six logical CPs.

MVS5’s weight gives it two physical CPs of service divided between its two logical CPs. Therefore, MVS5’s CPs appear to be faster than those of MVS2. It is like the old question: “Is it better to have a server with a large number of slower CPs, or one with a small number of faster ones?”

It is important to remember that all these considerations apply to processor, but also to zIIP and zAAP in an MVS LP. Therefore, in an LPAR profile the customer should define the number of logical zIIPs and zAAPs, together with their respective weights.
7.22 z196 PU pools

A pool is a set of non-shared identical personality physical PUs. With the z196, there are the following pools: CPs, IFLs, ICFs, zAAPs, and zIIPs.

New RMF reports show all LPs involved in all pools. The rules for managing pools are as follows:

- Logical PUs are dispatched from a supporting physical pool only (for example, logical CPs are dispatched from the CP pool only).
- The logical pool “width” (number of logical PUs per LP) limits the maximum number of shared logical PUs, when an LP is activated.
- A physical PU is placed in its respective pool by the following actions:
  - Activate (POR)
  - Concurrent Upgrade - On/Off Capacity on Demand (CoD), Capacity BackUp (CBU), Customer Initiated Upgrade (CIU), Capacity Upgrade on Demand (CUoD) MES
  - Dedicated LP deactivation
  - Dedicated LP configure logical PU OFF
- A PU is removed from its respective pools by the following actions:
  - Concurrent downgrade - On/Off CoD, CBU, PU conversion MES
  - Dedicated LP activation (“width” permitting)
– Dedicated LP configure logical PU ON ("width" permitting)

Target LPx or PU sharing is calculated from the right pool, and in the HMC there is a weight per each pool in the same LP.

Figure 7-22 on page 465 depicts the following:

On the z196:

– Target LPx (or LPAR Share) =
  \[
  \frac{\text{# Pool PUs} \times (\text{LPAR Pool Weight})}{\text{Total Pool Weight}}
  \]

• Cannot exceed the number of online logical PUs in the LPx
• There are separate LPAR weights for CP and zAAP, as shown in Figure 7-22. The weight for the CP is 250. The weight for the zAAP is 250 as well.

– Pool PUs (Physical): CP = 10, ICF = 1, IFL=2, zAAP=3
– Total Pool Weights: CP = 1000, zAAP = 1000, IFL= 400, ICF= 100
– For LP MVS1: 10 x (250/1000) = 2.5 CP share and 3 x (250/1000) = 0.75 zAAPs
– There is no interaction among engine weights from different pools.
7.23 Capping workloads

Capping workloads

*Capping* is used to limit, artificially, the PU consumption rate of a specific set of dispatchable units (as TCBs and SRBs), usually associated with user transactions workload.

**Major reasons for capping**

Here we discuss common reasons why people cap LPs. In certain cases, these may be common reasons; we do not necessarily agree that capping is a good way to achieve your objective in all these cases.

- The logical partition software products are charged on consumed MSU/h basis.
  
  Usually the consumed MSU/h per month is calculated by the peak month hour. To avoid a high peak many customers cap such a logical partition.

- Enforcing Service Bureau limits on customer’s load.
  
  With many businesses outsourcing their computing environments to Service Bureaus, using capping becomes an easy way to ensure that customers only get the CP resource they pay for.

- Reserving capacity for political reasons.
  
  Following an upgrade, some installations use capping to avoid a sudden significant improvement in response time, which will then evaporate as the additional capacity gets used up.
Avoid PUs running close to 100% to inhibit the increase of the processor time per transaction.

It is a known fact that with processors running close to 100%, there is a consistent increase in the transaction processor time. Capping the transactions less important to the business decreases the processor load and reverts the process.

WLM protecting discretionary workloads

This happens, when all the transactions with a numeric goal are happy. In this case these transactions are internally capped by WLM to let processor cycles to the discretionary transactions.

The following reasons are not described in Figure 7-23 on page 467, because they are not recommended.

You want to isolate test system usage from production usage

A test system is often placed on the same server as production systems. As with many production systems, their utilization depends on the time of day; some systems may be used for online processing, while other systems may be used for overnight batch processing. Any spare CP resource is required for the production systems. By LPAR capping the test system, you ensure that any available CP resource, above that guaranteed to the test LP, will be available for use by the production systems.

While this is a popular use of capping, in our opinion, if you specify the right weights for the test and production LPs, LPAR LIC will ensure that the production LP gets the capacity that has been guaranteed by its weight. Keep in mind that capping the test LP does not allow the production LP get more capacity if both LPs are 100% busy—all it does is stop the test LP from using unused cycles above its Target(LPx).

Some workloads are suspected to be in a loop.

If you have an application program that is apparently looping, but you do not want to cancel it right away (you are not sure) and you do not want this LP to consume more than its share of CP resources, you may use WLM Resource Group capping to limit the CP consumption by that transaction. However, you may shoot yourself in your foot because capping does not free the ENQ and lock resources owned by the suspicious task.
7.24 Types of capping

Types of capping in z196 workloads:

- **LPAR capping** (through LPAR using Weights or by decreasing the number of logical CPs), where the full LP is capped
- **WLM Resource Group capping** (through WLM), where just a set of service classes in the sysplex are capped
- **Soft capping** (through LPAR plus WLM), where the full LP is capped
- **Discretionary capping** (a hidden capping through WLM), where the happy service classes are capped

Types of capping

As indicated in Figure 7-24, the z196 plus z/OS system have the following types of capping:

- **LPAR capping**, where the full LP is capped. Refer to “LPAR capping” on page 470. The intelligence and the capping executor are located in the LPAR hypervisor:
  - The limit for capping is established by the logical partition weights (in HMC LPAR profile).
  - The limit of capping is determined by the number of logical PUs, when this number is less than the number of the non-shared physical PUs (from the same PU pool).
- **WLM Resource Group capping**, where just a set of service classes in the sysplex are capped. The limit is indicated through a Resource Group construct in the WLM policy. The intelligence and the executor are located in WLM.
- **Soft capping**, where the full LP is capped. The limit is indicated in MSU/h in the HMC LPAR profile. The intelligence is located in WLM and the executor is in the LPAR hypervisor.
- **Discretionary capping** (a hidden capping), where the happy service classes with numeric goals in the sysplex are capped. There is no external limit for such capping. The intelligence and the executor are located in WLM.
7.25 LPAR capping

- LPAR capping limits the PU consumption in an LP through its weights (in HMC LPAR profile)
- Without capping an LP can receive more CP service than its weight allows:
  - When other LPs are not using all of their share
  - When an LP is deactivated
  - This is generally a good thing - you make the best use of the available capacity
- The intelligence and the capping executor are located in the LPAR hypervisor
- The limit of capping is also determined by the number of logical PUs in the LP

**Figure 7-25  LPAR capping**

**LPAR capping**

LPAR capping is a function used to ensure that an LP’s use of the physical PU cannot exceed the amount specified in its Target(LPx). LPAR capping is set in the processor page for the LP in the HMC Image Profile.

Normally, an LP can get more PU resources than the amount guaranteed by its Target(LPx); in “LPAR weights” on page 462, we discuss how to calculate the Target(LPx) value. Usually, this is a good thing, because if there is spare PU resource that is not required by another LP, it makes sense to use it for an LP that needs more PU resource. This can happen when the other LPs are not using all their share or are not active—remember that when an LP is deactivated, the Target(LPx) of the remaining active LPs is recalculated.

If you want to prevent an LP from ever being able to use more than its Target(LPx), even if there is spare PU resource available, you would use the LPAR capping feature. LPAR capping is implemented by the LPAR hypervisor by observing the PU resource consumed by a logical PU in a capped LP, and acting if the utilization starts to exceed the logical CPs Target(LCPx).

At very frequent intervals (every few seconds or so), LPAR compares the Target(LCPx) of each logical PU of a capped LP to the amount of PU resource it has actually consumed over the last interval. Depending on the result of this comparison, LPAR LIC decides for what percentage of the time in the next interval that the logical PU should not be given access to a physical PU.
7.26 LPAR capped versus uncapped

Capped and uncapped LPs
Figure 7-26 illustrates what happens when there is spare CP resource and the server contains capped and uncapped LPs. Three LPs are shown, as explained here:

- MVS1 has a weight of 45 and is uncapped.
- MVS2 has a weight of 40 and is uncapped.
- MVS3 has a weight of 15 and is capped.

The total weight equals 100, and therefore each unit of weight is equal to 1% of server resource. For example, a weight of 45 equals 45% of the total server.

Each LP's weight is shown as a separate straight horizontal line in the chart. The total server utilization is shown as a separate line close to the top of the chart. This line never exceeds 98%, therefore indicating that there is always some spare server resource.

At one time or another, each LP requires less than its guaranteed share of CP resources. This spare capacity can be used by either of the uncapped LPs. When an LP uses more than its weight, its utilization line is above its weight line. This occurs for:

- MVS1 from time 60 to 150
- MVS2 from time 0 to 30
For MVS3, which is capped, its utilization line never extends above its weight line. It reaches its weight line and remains there even though there is spare CP resource, as shown by the total CP utilization line being below 100%. A performance impact would be noticed on MVS3 if it required more than its 15% of CP resources.

This completes the review of LPAR shared CP handling. You should now have sufficient knowledge of how LPAR works to be able to understand how the enhancements in WLM LPAR CPU Management are implemented.

**WLM resource group capping**

Be aware that WLM implements capping in its workloads as well, with the added flexibility that there are two Performance Group numbers in WLM; one for protection (smaller), and another for capping (larger), instead of just one as in LPAR capping.

Another advantage is granularity: in WLM capping, you can cap a set of service classes within a sysplex. With LPAR capping, you are capping the full LP.

A complete explanation of WLM Resource Group capping is beyond the scope of this book. Refer to *ABCs of z/OS System Programming Volume 11*, SG24-6327, for details about this subject.
7.27 Soft capping

- Sum of defined capacity as a basis for charge
- Ability to handle workload spikes

**Figure 7-27 Soft capping**

**Soft capping**

Workload charging (WLC) is a contract dealing with software fees. With WLC, the cost is based on the consumption of the LP running the product, measured in a 4-hour rolling average MSU/hour (captured along 5-minute intervals). Along this rolling average the installation can have peaks, beyond the agreed values, which may be balanced by valleys.

An installation can implement soft capping (also called *defined capacity*) to enforce a limit on LP consumption, and consequently to save on software fees.

Defined capacity is controlled by WLM and is executed by the LPAR hypervisor code. As illustrated in the example in Figure 7-27, the LPAR1 4-hour rolling average MSU/hour consumption is capped at 40 MSU/hour.
Group capacity in soft capping

Group capacity is an extension of soft capping in z/OS V1R8 that adds more flexibility; instead of soft capping each LP individually, an installation caps a group of LPs containing shared logical processors. WLM/LPAR management balances capacity (in MSU/h) among groups of LPs on the same CPC.

An installation defines a capacity group by entering the group name and group limit MSU/h 4-hour rolling average value (in HMC). Group capacity works together with individually-defined capacity LP capping, and an LP can have both a single capacity and a group capacity.

It is possible to define multiple groups on a CPC, but an LP can only belong to one group. A capacity group is independent of a sysplex and an IRD LPAR cluster. That is, in the same group, you can have z/OS from different sysplexes and different IRD LPAR clusters, but in the same CPC.

Each WLM in one LP manages itself independently from all other LPs, but it is informed about the consumption of all other LPs on the server. Each WLM calculates its defined share of the capacity group, based on the LP weight, as described here:

- This share is the target for the LP, if all LPs of the group want to use as much processor resources as possible.
- If one or more LPs do not use their processor resource share, this donated capacity is distributed over the LPs which need additional processor resource.
Even when an LP receives processor resource from another LP, it never violates its defined capacity limit (if one exists).

Looking at a server with three LPs (as portrayed in Figure 7-28 on page 474), based on the weights (465, 335, 200) and in the total server MSU capacity (400), the target MSU consumption for each LP is as follows:

- 186 MSU/h for LP A
- 134 MSU/h for LP B
- 80 MSU/h for LP C

LP A and LP B are not soft capped, and C is soft capped at 30 MSU/h. The three belong to a capacity group named Prod, which is capped at 200 MSU/h.

Here, we assume that the full server has processor capacity higher than 200 MSU/h, and that the number of logical processors in each LP is equal to the number of shared physical processors in the server.

If each LP takes all possible processor resource because the other two are idle, we may reach the maximum figures:

- LP A can use up to 200 MSU/h, limited by the group capping.
- LP B can use up to 200 MSU/h, limited by the group capping.
- LP C can use up to 30 MSU/h because an individual softcap is defined.

If all three LPs want to use as much resource as possible at the same time, we may reach the protection (guarantee) figures:

- LP A gets 99 MSU/h.
- LP B gets 71 MSU/h.
- LP C gets 30 MSU/h.
7.29 Intelligent resource director (IRD)

- Intelligent Resource Director is not actually a product or a system component; rather it is three separate but mutually supportive functions:
  - WLM LPAR CPU Management:
    - Vary Logical CPU Management
    - Vary Weight CPU Management
  - Dynamic Channel Path Management (DCM)
  - Channel Subsystem I/O Priority Queueing (CSS IOPQ)

Figure 7-29  IRD functions

Intelligent Resource Director (IRD)
Intelligent Resource Director was announced in October 2000 as one of the new capabilities available on the z196 and delivered as part of z/OS.

Intelligent Resource Director might be viewed as Stage 2 of Parallel Sysplex. Stage 1 provided facilities to let you share your data and workload across multiple system images. As a result, business transactions that supported data sharing could potentially run on any system in the sysplex, thus allowing WLM (for example) to move your workload to where more processing resources were available.

However, not all applications support data sharing, and there are many applications that have not been migrated to data sharing for various reasons. For these applications, IBM has provided Intelligent Resource Director, which basically gives you the ability to move the resource to where the workload is.

IRD support added
Intelligent Resource Director is implemented by the functions in:
- z/OS (in z/Architecture mode)
- Workload Manager (WLM) - see “Workload Manager (WLM) concepts” on page 482
- IBM System z machines
And by using existing function in the following system components:

- Hardware Configuration Dialog (HCD) - see “What is HCD” on page 492
- Dynamic I/O Reconfiguration
- I/O Supervisor (IOS)

**Intelligent Resource Director and WLM**

IRD uses facilities in z/OS Workload Manager (WLM), Parallel Sysplex, and PR/SM to help you derive greater value from your z196 investment. Compared to other platforms, z/OS with WLM already provides benefits from the ability to drive a server at 100% while still providing acceptable response times for your critical applications. IRD amplifies this advantage by helping you make sure that all those resources are being utilized by the right workloads, even if the workloads exist in different logical partitions (LPs).

**IRD functions**

As pictured in Figure 7-29 on page 476, IRD is not actually a product or a system component; rather it is three separate but mutually supportive functions:

- WLM LPAR CPU Management - see “WLM concepts” on page 482.
- Dynamic Channel Path Management (DCM) - see “Dynamic Channel Path Management (DCM)” on page 485.
- Channel Subsystem I/O Priority Queueing (CSS IOPQ) - see “Channel subsystem I/O priority queueing” on page 487.
7.30 WLM LPAR CPU management

There are two parts to WLM LPAR CPU management:
- WLM LPAR Weight management
  - Automatically changes the weight of a logical partition
  - Based on analysis of current workloads by WLM
- WLM Vary CPU management
  - Automatically varies a logical CP online or offline in an LP
  - Based on analysis and requests from WLM

Software managing hardware resources:
- Software - WLM Goal mode
- Hardware - Shared CPs and logical partition weights
- Parallel Sysplex - used to share WLM information between systems

WLM LPAR CPU management
WLM LPAR CPU management is a capability provided by z/OS. It is available on z196. It is one of the three components of Intelligent Resource Director.

Here we provide an introduction to this new capability, including information you need to decide if WLM LPAR CPU management is appropriate for your environment. If you determine that it is, the subsequent sections in this part provide the information to plan for, implement, and manage it.

WLM LPAR CPU management is implemented by WLM and LPAR hypervisor. WLM LPAR CPU Management, as shown in Figure 7-30, actually consists of two separate but complementary functions:
- WLM LPAR Weight management
  Its role is to dynamically change the LPAR weight of logical partitions to help a workload that is missing its goal.
- WLM LPAR Vary CPU management
  Its role is to dynamically change the number of online logical CPs in a logical partition (LP), to bring the number of logical CPs in line with the capacity required by the LP.
WLM LPAR CPU management
WLM LPAR CPU management is a new capability provided by z/OS. It is available on IBM System z and later servers when the z/OS LP is running in WLM Goal mode. It is one of the three components of Intelligent Resource Director.

Here we provide an introduction to this new capability, including information you need to decide if WLM LPAR CPU management is appropriate for your environment. If you determine that it is, the subsequent chapters in this part provide the information to plan for, implement, and manage it.

WLM LPAR CPU management is implemented by z/OS Workload Manager (WLM) goal mode and IBM System z PR/SM LPAR scheduler Licensed Internal Code (LIC). WLM LPAR CPU management, as shown in Figure 7.32, actually consists of two separate but complementary functions:

- WLM LPAR Weight management
  Its role is to dynamically change the LPAR weight of logical partitions to help a workload that is missing its goal. Refer to “Intelligent Resource Director benefits” on page 480 for more information.

- WLM LPAR Vary CPU management
  Its role is to dynamically change the number of online logical CPs in a logical partition (LP), to bring the number of logical CPs in line with the capacity required by the LP.

Parallel Sysplex
It is also necessary for the systems to be in a Parallel Sysplex, in order to share the WLM information between the systems.
7.31 Intelligent Resource Director benefits

Using WLM, PR/SM, and Parallel Sysplex, resources are dynamically moved to where they are needed.

**Example: Day shift**

- Partition 1: Weight = 75
- OLTP (Importance 1)
- Business Intel. (Importance 2)

**Example: Night shift**

- Partition 1: Weight = 25
- Batch (Importance 3)
- Business Intel. (Importance 2)

**IRD Vary Weight CPU management**

Figure 7-26 on page 471 provides a simplified view of what IRD does for you through the WLM Vary Weight CPU management function. It shows a CPC (server) with two LPs. One LP contains an OLTP workload, defined in WLM as having Importance 1, and a batch workload, defined in WLM as having Importance 3. The other LP contains a Business Intelligence workload that is defined to WLM as having Importance 2. Both the batch and the Business Intelligence workloads are capable of using the capacity of the whole server if allowed to.

To provide the OLTP workload with the resources it requires to meet its goals during the prime shift, Intelligent Resource Director sets the LPAR weight of that LP to 75. The weight of the Business Intelligence LP is set at 25. However, in the evening shift when the OLTP workload has gone away, Intelligent Resource Director will adjust the weights so that the Business Intelligence LP, which is of higher importance than batch, now gets 75% of the server, and the LP containing the batch workload now gets 25%.

You will also notice that during the prime shift, the OLTP DASD has more channels, whereas in the evening, there are more paths to the Business Intelligence DASD. Intelligent Resource Director has also automatically adjusted the channel configuration to provide more paths to the DASD subsystem serving the more important workload. This is an implementation of the IRD Dynamic Channel Path Management (DCM) function.
**Note:** z196 servers have introduced a new LPAR function named HiperDispatcher, whereby the LPAR and the z/OS dispatcher join efforts to improve the use of cache and TLBs in physical PUs. Refer to “Software/hardware cache optimization” on page 140 for more information about the HiperDispatcher function. This new function affects the IRD CPU Management function.
7.32 WLM concepts

**LPAR cluster**

The Intelligent Resource Director (IRD) extends the concept of goal-oriented resource management by allowing you to group system images that are resident on the same physical server running in LPAR mode, and in the same Parallel Sysplex, into an "LPAR cluster", as shown in Figure 7-32. This gives WLM the ability to manage CPC and channel subsystem resources, not just in a single image but across the entire cluster of system images. A CPC can have multiple LPAR clusters supporting different Parallel Sysplexes, and a Parallel Sysplex can in turn comprise multiple LPAR clusters in different CPCs.

**Workload Manager (WLM) concepts**

The following descriptions are provided for background information about WLM.

WLM is a z/OS component responsible for managing system resources in such a way that the workloads identified as being the most important will achieve their objectives. WLM provides the following functions:

- Simplicity, since goals are assigned in the same terms as in existing Service Level Agreements (instead of having to assign relative dispatching priorities in the IEAIPSxx), and the use of an ISPF/TSO application to define such goals.
- It is more closely linked to a business’s needs. Workloads are assigned goals (for example, a target average response time) and an importance. Importance represents how important it is to the business that the workload meet its goals. In compatibility mode, all workloads are defined in terms of relative dispatching priority to the other workloads—and
these dispatching priorities do not necessarily reflect the business importance of the associated workloads. Also, relative dispatching priorities mean that a workload may keep getting resource whenever it requests it, even if it is overachieving its goal while other workloads with a lower dispatching priority are missing their goals.

- WLM recognizes new transaction types, such as CICS, IMS DC, DDF, IWEB, UNIX System Services (USS), DB2 parallel query, MQSeries®, APPC and so on, allowing reporting and goal assignment for all of these workload types.

- It is particularly well-suited to a sysplex environment because WLM in Goal mode has knowledge of the system utilization and workload goal achievement across all the systems in the sysplex.

  **Note:** This cross-system knowledge and control has a much more significant impact in an IRD environment.

- It provides much better RMF reports, which are closely aligned to the workloads and their specified goals. For example, if you defined a goal that 80% of transactions should complete in 1 second, RMF will report the actual percent of transactions that completed within the target time.

- WLM is constantly adjusting to meet the goals of the *current* workload.

- It provides dynamic control of the number of MVS address spaces, such as:
  - Batch Initiators
  - HTTP servers

- WLM plays a central role in dynamic workload balancing among z/OS images in a Parallel Sysplex with data sharing. WLM works with VTAM Generic Resources, Sysplex Distributor for TCP/IP, CICS MRO, and IMS Shared Message Queues to route transactions to the most appropriate system.

- WLM provides the ability to decide which image a batch job can run in based on the availability of a real or abstract resource (using the WLM Scheduling Environment feature).

- WLM provides the possibility of specifying both a minimum amount of processor resource that a Service Class Period is guaranteed if it needs it, and a maximum amount of processor resource that a Service Class Period can consume.

**Workload Manager highlights**

WLM has two main sets of routines: Those that attempt to achieve transaction goals (these strive to make the system as responsive as possible), and those that attempt to maximize the utilization of the resources in the environment (these aim for maximum efficiency).

- The Policy Adjustment routine, enforcing transaction goals, consists of the following:
  - WLM may change the priority of given tasks, including the dispatching priority, the weight of an LP (a WLM LP CPU Management function), and the specification of a channel subsystem I/O priority.
    
    This function employs a donor/receiver approach, where an important workload that is missing its goal will receive additional resources—resources that are taken away from other workloads that are overachieving their targets or workloads and that are less important (as defined by the installation). One of the objectives of this function is that the workloads within a given importance level will all have a similar performance index (PI), which is a measure of how closely the workload is meeting its defined goal.
  - Server address space (AS) creation and destruction (AS server management routines).
- Providing information to the dynamic workload balancing functions, such as VTAM Generic Resources or Sysplex Distributor, to help them decide on the best place to run a transaction.

- Resource adjustment routines, enforcing maximum utilization of the resources.

These are designed to maximize the throughput and efficiency of the system. An example would be the new WLM LPAR Vary CPU Management function, which will vary logical CPs online and offline in an attempt to balance required capacity with LPAR overhead.

There are several types of goals:

- Average response time (1 second, for example)
- Percentile response time (80% of the transactions with response time less than 2 seconds)
- Execution velocity, which is a measure of the amount of time the workload is delayed waiting for a resource that WLM controls

To provide this information, WLM tracks transaction delays, such as:

- Processor delay
- Storage delay (page faults and swapping)
- I/O delay
- AS Server queue delay

The following are some reasons why an important goal might not be reached and how WLM adjusts the way it manages system resources to compensate:

- Processor delay - the dispatching priority or LPAR weights are increased.
- Storage delay - storage isolation figures or swapping target multiprogramming level (TMPL) or system think time are raised.
- I/O delay - the I/O priority is increased in the UCB, channel subsystem and 2105 control unit queues, or an additional alias may be assigned to the device for devices that support Parallel Access Volumes.
- AS Server queue delay - a new server address space is created.

For the sake of clarity, we wish to point out that the System Resource Manager (SRM) component of z/OS still exists. However, for the sake of simplicity, we do not differentiate between the two in this book. While a particular action may be carried out by either SRM or WLM, we always use the term WLM.

**Note:** For information about options, problems or prerequisites for WLM CPU Management, refer *z/OS Intelligent Resource Director*, SG24-5952. For more information about WLM, refer to *ABCs of z/OS System Programming Volume 11*, SG24-6327.
7.33 Dynamic Channel Path Management (DCM)

DCM objectives

- Improve overall I/O performance
- Simplify the I/O configuration definition task
- Reduce skills required to manage z/OS
- Maximize the utilization of installed hardware
- Enhance availability
- Reduce the need for more than 256 channels

Dynamic Channel Path Management (DCM)

DCM is a new capability, designed to dynamically adjust the channel configuration in response to shifting workload patterns. It is a function in IRD, together with WLM LPAR CPU Management and Channel Subsystem I/O Priority Queueing.

Note: Prior to Dynamic Channel Path Management, all channel paths to I/O control units had to be statically defined. In the event of a significant shift in workload, the channel path definitions would have to be reevaluated, manually updated via HCD, and activated or PORed into the configuration.

Improve performance

DCM can provide improved performance by dynamically moving the available channel bandwidth to where it is most needed. Prior to DCM, you had to manually balance your available channels across your I/O devices, trying to provide sufficient paths to handle the average load on every controller. This means that at any one time, some controllers probably have more I/O paths available than they need, while other controllers possibly have too few.

DCM attempts to balance the responsiveness of the available channels by moving channels to the controllers that require additional bandwidth, even when the system is in WLM Compatibility mode.
Simplify I/O definitions
Because DCM can dynamically move I/O paths to the LCUs that are experiencing channel delays, you can reduce the CU/channel-level capacity planning and balancing activity that was necessary prior to DCM. Using DCM, you are only required to define a minimum of one non-managed path and up to seven managed paths to each controller (although a realistic minimum of at least two non-managed paths are recommended), with DCM taking responsibility for adding additional paths as required, and ensuring that the paths meet the objectives of availability and performance.

Skill to manage z/OS
It is still necessary to understand the performance and availability characteristics of all your installed hardware. DCM can only work within the confines of the physical connectivity that you design and implement. However, to the extent that DCM is self-tuning, it should be possible to spend considerably less time on configuration management and monitoring and capacity planning. This allows you to free up scarce technical resources to work on other more valuable tasks.

Maximize use of installed hardware
As one part of the continuing drive to reduce the cost of computing for the zSeries and z9 platform, DCM can help you drive more traffic to your DASD subsystems without necessarily having to invest in additional channels. DCM may let you increase the overall average utilization of your channels, without adversely impacting the response time for the connected subsystems.

Enhance availability
When attaching a DASD subsystem to a z196 CPC, there are a number of things to take into account in order to achieve maximum availability and performance from the configuration. Within the CPC, there are a number of redundant components provided (channel cards, STIs, MBAs, and so on). Ideally, you would attach a control unit to channels that are spread over as many of these components as possible, minimizing the number of single points of failure.

Reduce the need for more channels
DCM can reduce the need for more channels by configuring every channel with the maximum number of devices, while still delivering acceptable performance across all the control units. Because DCM dynamically adjusts to changing workloads, if you happen to end up with multiple busy control units on a channel (a situation that can easily happen today, and that will result in degraded performance), DCM will react to the resulting increased Pending time (or *Pend time*) by adding idle or less busy channels to the control units that are suffering high Pend times.

Note: DCM is implemented by exploiting existing functions in software components (in z/OS), such as WLM, IOS, HCD, Dynamic I/O Reconfiguration; and in hardware components, such as an IBM System z196, ESCON Directors, FICON Directors and DASD controllers.

DCM provides the ability to have the system automatically manage the number of ESCON and FICON channels.
7.34 Channel subsystem I/O priority queueing

Channel Subsystem I/O Priority Queuing
Channel Subsystem I/O Priority Queuing is a capability delivered on z196 CPCs, and exploited by z/OS. It is the third component of Intelligent Resource Director.

Channel Subsystem Priority Queuing is an extension of the existing concept of I/O priority queuing. Previously, I/O requests were handled by SAP on a first-in, first-out basis. This could, at times, cause high priority work to be delayed behind low priority work.

With Channel Subsystem Priority Queuing, if important work is missing its goals due to I/O contention on channels shared with other work, it will be given a higher channel subsystem I/O priority than the less important work. This function goes hand in hand with the Dynamic Channel Path Management, as additional channel paths are moved to control units to help an important workload meet goals. Channel Subsystem Priority Queuing ensures that the important workload receives the additional bandwidth before less important workloads that happen to be using the same channel.

Workloads and LPs
z/OS in WLM uses this new function to dynamically manage the channel subsystem priority of I/O operations for given workloads based on the performance goals for these workloads as specified in the WLM policy.

In addition, because Channel Subsystem I/O Priority queueing works at the channel subsystem level, and therefore affects every I/O request (for every device, from every LP) on
the CPC, you can also specify a single channel subsystem I/O priority that is to be used for all I/O requests from systems that do not actively exploit Channel Subsystem I/O Priority Queueing.

**I/O requests per channel prioritization**

Channel Subsystem I/O Priority Queuing applies to I/O requests to every type of device that is attached to the system, even for those attached via the old parallel channels. Channel Subsystem I/O Priority Queueing addresses the most significant remaining queuing points that up to now did not support the prioritization of I/O requests.

The I/O priority used for the UCB queue and the control unit internal queues is calculated based on understanding which service class periods (SCPs) in the system are competing for the same _device_. When WLM changes an I/O priority, it assesses the impact of the change only on systems that are in the same sysplex and competing for the same resource.

However, neither of these functions has any effect on how I/O requests are handled when they arrive at SAP coming from different LPs in the CPC. If you consider that a large modern CPC is capable of driving many thousands of I/O requests per second, that results in a significant number of requests hitting the channel subsystem (SAP).

Some of those requests will be from high importance workloads, and some will be from lower importance workloads. Additionally, in an LPAR environment, some of the LPs may be running production workloads, while other LPs may be running test or development (lower importance) workloads.

Prior to the IBM zSeries, there was no way, at the channel subsystem level, to differentiate between all these requests, meaning that high importance I/O requests may be delayed behind other, lower importance requests. And as time goes on, and the number of supported logical partitions per CPC and the processing power of the CPC increases, this situation would have become even more critical if it had not been addressed. Channel Subsystem I/O Priority queuing gives you the ability to differentiate between these requests at the channel subsystem level.

**Waiting for an SAP and channel**

Prior to Channel Subsystem I/O Priority queuing, requests that were queued waiting for an SAP or waiting for an available channel were handled in a first-in, first-out (FIFO) manner. There was no discrimination between I/Os from discretionary workloads and production online workloads, or between I/Os coming from the systems programmer sandbox LP and a production LP.

**DCM and channel bandwidth**

In addition to giving you more control over the order in which I/O requests are processed, Channel Subsystem I/O Priority Queuing is designed to be supportive of Dynamic Channel-path Management (DCM). DCM allows the channel bandwidth provided to control units to be managed based on the WLM goals of the SCPs using the control unit.

However, adding bandwidth to a given control unit provides more resources to _all_ work using that control unit. If both high importance and low importance SCPs are using the same control unit, a large amount of bandwidth might need to be provided to the control unit to ensure the high importance work meets its goal, thus giving a free ride to the low importance work.
Channel Subsystem I/O Priority Queuing ensures that the added bandwidth benefits the higher importance workloads first, by prioritizing those I/O requests to be selected by the channel subsystem ahead of the requests from the less important SCP. Giving the high importance SCP a higher channel subsystem priority minimizes the bandwidth required by the high importance SCP to meet its goals, since the high importance SCP will not have to queue behind the requests from the low importance SCP.

For more information about this topic, refer to z/OS Intelligent Resource Director, SG24-5952.
Hardware Configuration Definition (HCD)

Several hardware components, including PR/SM and the channel subsystem (CSS) together with the IBM z/OS operating system need to know what hardware resources (mainly I/O) and other software information are available in the system and how these resources are connected. This information is called the configuration and it is provided by your installation. You must describe this configuration to be used at IPL by z/OS (software) and to be used at power-on reset by the channel subsystem and PR/SM (hardware). In mainframes is not possible to have the concept of “plug and play” (where the operating system discovers such configuration) due to system complexity and security issues.

Hardware Configuration Definition (HCD) is a z/OS component used by your installation to describe your hardware resources such as: servers, logical channel subsystems, logical partitions, channels, switches, controllers, devices and some software informations. This is done through a TSO interactive end-user interface. Then, HCD needs a running z/OS system. A strong point about HCD is the validation checking that it does as you enter configuration data, helping to eliminate errors before you attempt to use the I/O configuration.

HCD consolidates all this configuration information (that could span more than one server) - to be used by hardware and software - in one output z/OS data set named IODF. Then, an IODF is used to describe multiple hardware and software configurations.

A configuration consists mainly of I/O hardware resources available to the operating system and the connections between these resources. When you define a configuration, you need to provide both physical and logical information about these resources. For example, when defining a device you provide physical information, such as its type and model, as well as logical information such as the identifier you assign in the configuration definition.

Many of the concepts mentioned in this chapter are covered more deeply in chapters Chapter 3, “Introducing the IBM zEnterprise” on page 165, and Chapter 4, “zEnterprise BladeCenter Extension Model 002 (zBX)” on page 301 of this volume 10 and it is truly recommended the reading in advance of such chapters.
8.1 What is HCD

- Combines hardware (IOCP) and software (MVSCP) definitions into a single process
- Allows multiple software and processor configurations to be defined in a single database
- Creates a single definition for each physical control unit and device
- Provides immediate online validation of configuration data
- Contains extensive online help
- Provides dynamic I/O reconfiguration

Figure 8-1  HCD characteristics

Before HCD
Before HCD became available, you had to use I/O configuration program (IOCP) to define the configuration to the channel subsystem at power-on reset (POR) and use the z/OS Configuration Program (MVSCP) to define the configuration to the z/OS operating system. With MVSCP and IOCP, you were limited to defining one server or one operating system per input data set. This meant that you needed more than one input data set when you used MVSCP or IOCP.

Multiple server configurations
The configuration you define with HCD (a z/OS component) may consist of multiple servers with multiple logical channel subsystems (LCSS), each containing multiple logical partitions with multiple channels, switches, I/O control units and devices. HCD stores the entire configuration data in a central repository (a z/OS data set), called the input/output definition file (IODF).

The IODF, as single source for all hardware and software definitions for a multi-server, or multi-logical partition complex, eliminates the need to maintain several independent MVSCP or IOCP data sets. This means that you enter the information only once, using an interactive dialog.
Single definition

When you define a configuration, you need to provide both physical and logical information about these resources. For example, when defining a device you provide physical information, such as its type and model, as well as logical information such as device number (identifier), or if its UCB should be located below or above the 16-M virtual address.

HCD provides an interactive interface that allows you to define the hardware configuration to both the channel subsystem and the operating system. The hardware information generated by the HCD is used by PR/SM and the channel subsystem (SAPs and channels)—and not by the I/O control units.

Online validation of data

HCD consolidates the hardware and software I/O configuration processes under a single interactive end-user interface. The validation checking that HCD does, as you enter data, helps to eliminate errors before you attempt to use the I/O configuration.

Online help

HCD offers an extensive help facility. From any panel, you can get context-sensitive help by pressing the F1=Help key.

Dynamic I/O reconfiguration

HCD, combined with the Dynamic I/O Reconfiguration Management function—mainly located in the input output supervisor (IOS), a z/OS component—allows you to make changes to either the hardware or software configurations without the need of a POR or an IPL. This greatly enhances overall system availability by making possible the elimination of scheduled outages that were previously necessary when parts of the configuration were changed.

Note: This chapter on HCD illustrates the many panels used in creating a new configuration. It is not a complete definition and does not show all of the panels and the various options needed. It is an attempt to familiarize you with the concepts available with HCD.

See z/OS Hardware Configuration Definition: User's Guide, SC33-7988, and z/OS Hardware Configuration Definition Planning, GA22-7525, for detailed information about using HCD.
8.2 IOCP example

Figure 8-2  IOCP example

IOCP example

Figure 8-2 shows a portion of an IOCP definition that describes one z-109 server system element, as shown in Figure 8-3 on page 495, which illustrates the relationship between LPs, LCSSs, CHPIDs, and PCHIDs. This illustration also includes the I/O devices and switches used in the IOCP example shown in Figure 8-2. Even though, with HCD, you do not need to write the IOCP statements, these statements are optionally produced by HCD based on how you fill out the HCD panels.

This IOCP definition shows the implementation of logical channel subsystems (LCSS). They are needed because existing software works with single-byte CHPID numbers, producing the limitation of 256 CHPIDs. The new architecture in the z9 EC and z990 provides multiple sets of channel definitions named LCSSs, each with a maximum of 256 channels. Existing operating systems would be associated with just one LCSS and work with a maximum of 256 CHPIDs. Different LPs can be associated with different LCSS definitions.

Thus, a single operating system instance (using existing code) still has a maximum of 256 CHPIDs, but the system as a whole can have more than 256 CHPIDs. Then it is necessary to have multiple operating images (in multiple LPARs) to exploit more than 256 channels, but this is a common mode of operation in customer installations. Refer to “Logical channel subsystem (LCSS)” on page 249.
8.3 IOCP elements

The partial IOCP file shown in Figure 8-2 on page 494, shows the elements of a z9 EC configuration in this IOCP example, so let's take a closer look at some of them now.

- **LCSS definitions**
  The LCSSs used in the IOCDS created from this IOCP are stated in the RESOURCE statement. In the example, here we define two LCSSs and indicate which LPs are associated with each one. This defines the LCSSs without any need of a hardware install procedure.

- **Subchannel maximums**
  The MAXDEV parameter (in the RESOURCE statement) affects the HSA storage created at power-on-reset (POR). It specifies the maximum number of subchannels (device numbers) that can be defined in a logical partition of an LCSS. The MAXDEV value includes the devices you have defined, plus whatever expansion number you need through a dynamic I/O reconfiguration. Before the z9 EC, the maximum number of subchannels (UCWs) allowed was almost 64K per LP. With the z9 EC, this number almost doubled by the introduction of two sets of UCWS: the CSS0 with almost 64K and CSS1 with 64K. The CSS1 is used by z/OS to map the alias addresses generated by the parallel access volume (PAV) feature in the DASD controllers, as DS8000.
- **LCSS number for CHPID**
  
  Each CHPID statement specifies its LCSS number as part of the PATH parameter for unique identification purposes.

- **PCHID number**
  
  Each CHPID statement *must* include a PCHID parameter to associate the CHPID with a physical channel identification. The PCHID parameters can be added to the IOCP definitions by the CHPID mapping tool, or through HCD definitions, or defined in a standalone IOCP file, but the PCHID parameters must be present in all CHPID statements when the IOCDS is created.

- **LCSS number for PATH and LINK parameters**
  
  PATH and LINK parameters in CNTLUNIT statements must indicate the LCSS number associated with each path and link.
8.4 Hardware and software configuration

![Diagram of hardware and software configuration flowchart]

**IODF and IOCDS data**
An IOCDS is located in the hard disk of the service element (laptop). It is created by IOCP reading the z/OS data set named input/output definition file (IODF) created by HCD. A major difference between these data sets is that the IOCDS contains the configuration for a specific server, while the IODF contains configuration data for multiple servers. IODF is required when IPLing an LP.

**System programmer interactions with HCD**
Here we cover the steps for installing and initializing an IBM System z server.

- The system programmer, interactively with HCD panels, creates the IODF Production data set. This data set contains the description of all the servers (in an HMCplex, which consists of all the Service Elements (laptops) connected by a token ring LAN), including Logical Channel Subsystems (LCSS), LPs, channels, switches, I/O control units, devices, and software information (used at IPL). Note that MBAs and STIs are not described here.
- IOCP program - The next step is executed by the IOCP program, a z/OS batch component, which can be invoked internally by HCD itself. IOCP may read the entire configuration describing all the servers in the HMCPlex. This data is passed to the Service Elements (SE). The configurations referring to other servers are sent by the SE through the LAN to the respective SEs. The local ones are stored in the local IOCDS, just a file in the hard disk laptop.
Power-on reset (POR) - Next is the power-on reset (POR), an action started by the operator through an icon of the hardware management console (HMC). In reality, HMC is the name of the software running in the desktop connected by a LAN with the SEs of the HMCplex. Through these desktops, the operator staff has a focal point to operate all the machines.

POR does many things, such as:

- Loading the microcode in the control storage of all PUs in the server.

Hardware system area (HSA) is allocated in the lower portion of book 0 in central storage. HSA is a piece of central storage that does not belong to any LP. It contains, after the POR:

- Millicode (a sort of subroutine from the microcode)
- PR/SM (LPAR) object code
- Copy of CFCC object code, to be copied to each Coupling Facility LP
- LP definitions from IOCDS
- I/O configuration data from IOCDS, constituted mainly by UCWs organized in one UCW table per LP. When you dynamically activate a new configuration, HSA is updated to reflect the new hardware I/O configuration definition.

If you plan to use dynamic I/O reconfiguration to add equipment to your configuration, you must specify an HSA size big enough to contain the new information. It is done by defining in HCD/IOCP the MAXDEV option, which tells how many devices will be supported. Refer to 8.2, “IOCP example” on page 494 for an example of the MAXDEV option.

Logical partition activation - At POR completion, all LPs are defined and the entire I/O configuration is described in the HSA. The next step is to activate the LPs. In the HMC console panels, the installation can define the LPAR profile, where the attributes of each LP are defined, such as the number of logical CPUs, weight, central storage size, capping, and so on. For the LP activation there is no need to access the IODF data set.

Initial programming load (IPL) - The IODF production data set is read by the z/OS IPL program, which reads the I/O configuration referring to the IPLed LP. For each device, a UCB is created mainly in ESQA, together with the Eligible Device Table (EDT). This table associates a set of devices with an esoteric name and the device preference list. This list creates a sequenced priority of device types, to be used if you are allocating a new data set and you have more than one possible device type as a candidate. Refer to “EDT and esoterics” on page 514.
8.5 HCD functions

Figure 8-5 summarizes what you can do with HCD and how you can work with it.

**Single point of control**

With HCD you have a single source, the IODF production data set, for your configuration data. This means that hardware and software definitions as well as FICON director (switch) definitions and matrix connections can be done from HCD and can be activated with the data stored in the IODF.

**Increased system availability**

HCD checks the configuration data when it is entered, thereby reducing the chance of unplanned system outages due to inconsistent definitions.

**Changing hardware definitions dynamically**

HCD offers dynamic I/O reconfiguration management. This function allows you to change your hardware and software definitions on the fly. You can add devices or change devices, channel paths, and control units, without performing a POR or an IPL. You may also perform software-only changes, even if the associated hardware is not installed. However, to add a new LP or a new LCSS are not allowed. Refer to 8.6, “Dynamic I/O reconfiguration” on page 501.
Sysplex-wide activate
HCD offers you a single point of control for systems in a sysplex. You can dynamically activate the hardware and software configuration changes for systems defined in a sysplex or even out of it, if the servers are in the same HMCplex.

Accurate configuration documentation
The actual configuration definitions for one or more servers in the IODF are the basis for the reports you can produce with HCD. This means that the reports are accurate and reflect the up-to-date definition of your configuration. HCD provides a number of textual reports and graphical reports that can be either printed or displayed. The printed output can be used for documentation purposes, thus providing the base for further configuration planning tasks. The display function allows you to get a quick overview of your logical hardware configuration.

HCD also produces a PDS member with its view of the current configuration. Through a z/OS command, the installation can ask z/OS to compare and display the deviations from reality.

There is a free software product called hardware configuration management (HCM) where you enter your configuration graphically and HCM transforms it in an output to HCD. Refer to 8.62, “Hardware Configuration Manager (HCM)” on page 576.

Guidance through an interactive interface
HCD provides an interactive user interface, based on ISPF/TSO, that supports both the hardware and the software configuration definition functions. The primary way of defining the configuration is through the ISPF dialog. HCD consists of a series of panels that guide you through all aspects of the configuration task. The configuration data is presented in lists.

HCD also offers extensive online help and prompting facilities. Help includes information about panels, commands, data displayed, available actions, and context-sensitive help for input fields. A fast path for experienced users is also supported.

Batch utilities
In addition to the interactive interface, HCD also offers a number of batch utilities. You can use these utilities, for instance, to migrate your existing configuration data, to maintain the IODF, or to print configuration reports.

Cross-operating system support
HCD allows you to define both MVS-type (for example, z/OS) and VM-type configurations from z/OS.

LDAP interface capability
HCD provides search and update capabilities for IODF data via a Lightweight Directory Access Protocol (LDAP) interface. LDAP is an Internet protocol standard based on TCP/IP. LDAP is a protocol that makes directory information accessible. New entries can be added, existing entries can be altered or deleted, and it is possible to search for all matching entries using wildcards.
8.6 Dynamic I/O reconfiguration

Dynamic I/O reconfiguration is the ability to select a new I/O configuration definition without having to perform a power-on reset on the hardware or an IPL of the z/OS system.

It allows an installation to add, delete, or modify the definitions of channel paths, control units, and I/O devices to the software and hardware configurations. You can change the I/O configuration definitions for both software and hardware or for software only. It also allows you to dynamically change the EDT and the Device Preference List. To use dynamic I/O reconfiguration, HCD must be used to define the new configuration. However, with dynamic I/O reconfiguration you cannot add a logical channel subsystem or a new logical partition.

To implement a dynamic I/O reconfiguration you need to perform the following steps:

1. Define an IODF production with the full new configuration, including the devices that will not be changed, the new devices, and the devices to be changed. The devices to be deleted are omitted.

2. Issue the ACTIVATE command through the z/OS console or HCD panel in one z/OS image. Usually, changing the I/O hardware configuration should affect all the LPs of a server. Then in one of the LPs you need a dynamic reconfiguration for hardware and software, and in the others for software only. The z/OS where the hardware and software reconfiguration was ordered reads the new configuration and generates a delta list that includes the new elements and the ones to be deleted. This z/OS then synchronously updates the software configuration (UCBs) and the hardware configuration (UCWs), using...
a special instruction that is able to update the HSA. Also, a configuration token is kept in HSA to uniquely identify such a configuration.

- Software-only changes and configuration token - These changes must be done using the same IODF that was used to define the current hardware configuration in all the other LPs. You can use the software-only change to synchronize the software definitions in the other logical partitions with the hardware definition so that a subsequent hardware and software change can be activated. When a software-only change executes, the token in the IODF must match the current server configuration token in HSA for integrity purposes.

**Dynamic I/O reconfiguration benefits**

Dynamic I/O reconfiguration has the following benefits:

- It increases system availability by allowing you to change the I/O configuration while z/OS is running, thus eliminating the POR and IPL for selecting a new or updated I/O configuration.
- It allows you to make I/O configuration changes when your installation needs them, rather than waiting for a scheduled outage to make changes.
- It minimizes the need to logically define hardware devices that do not physically exist in a configuration (also known as over-defining a configuration).

If you have installed z/OS 1.4 (without the HCD exploitation facility) or an older release, the hardware dynamic reconfiguration is only done in LCSS 0, because such releases of z/OS do not support more than one LCSS.
8.7 Dynamic I/O reconfiguration device types

- **Static**
  - UIM does not support Dynamic I/O changes
- **Installation-static**
  - UIM supports Dynamic I/O
    - Specified with DYNAMIC=NO in HCD
- **Dynamic**
  - UIM supports Dynamic I/O
    - Specify DYNAMIC=YES in HCD

**Dynamic reconfiguration management**
This section introduces and explains a number of dynamic reconfiguration management concepts. These concepts and terms are used in subsequent sections of this chapter when discussing dynamic I/O reconfiguration.

**Unit Information Module (UIM)**
IOS does not understand the peculiarities of each possible device type. Each device type that exists has an associated software routine called the unit information module (UIM), written by the manufacturer, following a standard API, which is included in the IOS code. UIM contains the device support code. UIMs are invoked by IOS at device initialization and by HCD for testing the HCD definition of a specific I/O configuration that you may have. One of the functions of UIM is to specify whether or not the device type supports dynamic I/O configuration.

**Device types**
IOS treats device control information (for example, UCBs) differently depending upon whether the device can be dynamically reconfigured or not. This state may also be declared in HCD or through the DYNAMIC parameter at IODEVICE in IOCP. The possible device types are:

- **Static**
  A static device is a device whose type as defined in its UIM does not support dynamic I/O reconfiguration (usually an old device type). A static device cannot be dynamically added, deleted, or modified in the software configuration definition. Therefore, the device is not available...
for use until the next IPL of z/OS. However, the device can be added, deleted, and modified in the hardware configuration definition, but its channel path information cannot be modified.

**Installation-static**

This is a device whose type as specified in its UIM supports dynamic I/O configuration, but which you have defined as DYNAMIC=NO (default is NO) in the HCD definition. You might specify DYNAMIC=NO if your installation has old software such as customer programs and supplier programs and products that depend on specific device-related data structures such as UCB and EDT, or use existing old z/OS services that access these data structures, and are unprepared to handle dynamic changes to these structures.

Installation-static devices can be dynamically added to the software and hardware I/O configuration, but cannot be deleted or modified while z/OS is running. Devices can be dynamically added, deleted, and modified in the hardware configuration definition, but their channel path information cannot be modified. Usually the UCBs of such devices are located in SQA below the line. The control information for installation-static devices can be accessed using either old UCB services or the new ones. Defining devices as installation-static should be considered an interim measure until all the programs accessing their UCBs have been converted to the new UCB services.

Modifying or deleting an installation-static device requires two separate dynamic I/O reconfigurations.

Firstly, change the installation-static device to dynamic by activating a new IODF that defines the device as dynamic. No other device characteristics for that device can be changed on the redefinition activation.

Secondly, delete or modify the dynamic device (by activating another IODF that contains the appropriate changes to the device).

**Dynamic**

This is a device whose device type as defined in its UIM supports dynamic I/O configuration (in software and hardware). In order to dynamically add, modify, or delete such a device while z/OS is running, it is also necessary that the device be defined as dynamic in HCD (DYNAMIC=YES). Their UCBs reside in ESQA.

**Eligible device table (EDT)**

During dynamic I/O reconfiguration, it is possible to also change the EDT. Usually, z/OS uses one EDT to process allocation requests. However, when you dynamically change your system's software I/O configuration definition, z/OS may have to use two EDTs to process the change:

- The primary EDT processes all current and new allocation requests. z/OS runs with only the primary EDT until you make a dynamic I/O configuration change. z/OS activates a new primary EDT for the new I/O configuration, which makes the former primary EDT the secondary EDT.
- The secondary EDT receives no new allocation requests. The system deletes the secondary EDT when it finishes the allocation requests issued before the dynamic I/O change.
8.8 IODF data set

I/O definition file (IODF) data set
The I/O definition file (IODF) is a VSAM linear data set that is built and maintained by HCD. There are two types of IODF data set: the work IODF file, and the production IODF.

Work IODF file
A work IODF allows you to create a new I/O configuration definition or modify an existing one. You can have multiple work IODFs, each with a unique name, and all work IODF names must conform to the following convention:

'hhhhhhhh.IODFxx.xyyyyyyyy yyyy yyyy'

Where:

hhhhhhhh  High level qualifier—up to eight characters
xx  Hexadecimal number in the range of 00 through FF as the IODF suffix
yyyyyyyy  Optional qualifiers—up to eight characters each

HCD provides considerable flexibility in the choice of work IODF names. It is, however, recommended that you be more stringent in your naming conventions. It is recommended that you use the following convention when choosing a name for an IODF so in this manner you can easily ascertain the type of the IODF:

'hhhhhhhh.IODFxx.WORK'

'hshhhhhhhh.IODFxx'
**Production IODF**

A production IODF is the data set used to obtain the definitions needed to run the system. A production IODF is a read-only data set, which preserves the integrity of the IODF.

The naming convention of the production IODF is more restrictive. If the IODF is to be used for IPL and dynamic activation, the production IODF must have the following format:

'hhhhhhhh.IODFxx'

The production IODF is the copy used at IPL, and as an input to IOCP. It is created from the work IODF. During the IPL, the information in the production IODF is used to build the UCBs of the devices accessed by that LP. At IPL the device number of the device where the IODF is located is specified in the `load parm` at the hardware management console. In the LOADxx PARMLIB member (used by the IPL process) you specify the data set name IODF identification, such as:

- **xx:** The suffix
- **hhhhhhhh** The high-level qualifier

The following naming convention is a useful way to relate the current production IODF to its work IODF, and is easily extendable as new IODF versions are created:

- **ACTIVE PRODUCTION IODF:** 'hhhhhhhh.IODF30'
- **SUBSEQUENT WORK IODF:** 'hhhhhhhh.IODF31.WORK'
- **RESULTING NEW PRODUCTION IODF:** 'hhhhhhhh.IODF31'

**Catalog considerations**

In a multisystem environment, a single IODF can define several software (images) and hardware (servers) configurations. Observe also that the IODF data set must be z/OS-cataloged. In a scenario, where you wish to share an IODF data set among multiple z/OSs and each system is using a separate master catalog, you must define (in the master catalog of each system) an alias that relates to the user catalog on DASD that is shared among the systems. Define aliases and the user catalog before using HCD to define IODF data sets.

**Production IODF backup**

It is suggested that you maintain a backup copy of your production IODF on a separate volume that is accessible to all systems that are sharing the production IODF. When the primary IODF volume is inaccessible or the IODF itself is corrupted, the system can be IPLed through a backup IODF on the alternate IODF volume.

**IODF placement**

The production IODF to be used during IPL processing must be placed on the IODF device, pointed to by the load parm (L2) value on the IPL panel of your hardware management console. In an SMS environment, care should be taken to ensure that either:

- If the production IODF data set is not managed by SMS (there is not a Storage Class associated with the IODF), you might specify the IODF volume serial number when creating a production IODF.
- If the production IODF is SMS-managed, then the ACS routines are set up to automatically place the production IODFs on the IODF volume.

**Multiple configurations in a single IODF**

You can decide whether to create one IODF for each server, or to combine the I/O definitions for two or more servers in a single IODF. The advantage of including the definitions of two or more servers with shared devices (among the LPs in such servers) in one IODF is that all the
information is centralized in just one place; as a result, there is less chance of error, more control, and the data for shared devices has to be entered only once.

A second reason for considering maintaining configuration definitions for several servers in one IODF is that it allows you to easily move configurations from one LP to another (in either the same server or separate servers).

The general recommendation is to create one IODF for a logical server complex; that is, a group of servers that share I/O devices, as a sysplex. In this way, one z/OS system can be used for updating the IODF with the configuration changes, which minimizes the chance of error.

If you have previously had multiple IODFs defining your complex, the HCD copy/merge function can be used to copy relevant parts of configuration data from a source to a target IODF, or even from areas within an IODF.
8.9 Definition order

[BC][BC][BC][BC][BC]

Recommended definition order within HCD

- Operating system
  - EDTs
  - Esoterics
- Switches
  - Ports
- Processor
  - Partitions (LPAR only)
  - Channels (chpids)
- Control units
- Devices
- Consoles

Figure 8-9 Definition order

Definition order

You can define the objects of a configuration in almost any order, but at one point you have to connect objects together. You can only connect objects that are already defined; therefore, it is useful to define the objects in a logical order. For example, when defining I/O devices during the hardware definition, you are prompted to add devices to existing operating system definitions. Therefore, it is useful to define the operating system before defining the devices.

The suggested sequence for defining a configuration is shown in Figure 8-9 and is as follows:

- Operating systems
- EDTs (z/OS-type only)
- Esoterics (z/OS-type only)
- Switches
- Servers
- Logical Channel Subsystems
- Logical Partitions
- Channel paths
- Control units
- Devices
- Consoles
8.10 HCD primary menu

On entering HCD, you are presented with the HCD Primary Menu panel. HCD provides an action bar-driven interface that exploits many of the usability features of the Common User Access (CUA) interface. To select an item from a numbered selection list, type the number you want to select in the input field (left of the first list item) and press Enter.

The first time you use HCD, you must enter the IODF data set name that you wish to use. If this is not the name of an existing IODF, HCD creates a new work IODF for you.

An example of a numbered list is the HCD primary task selection panel, shown in Figure 8-10. This panel is displayed when you start an HCD session.

To create a new IODF, specify a work IODF (choosing a name) as follows:

```
I/O definition file . . . 'ROGERS.IODF05'
```

Then, enter 1 in the input field and press Enter.

If you have a production IODF, it is advisable to start from the active production IODF as a base. This is mandatory for a dynamic environment. Enter the new IODF name enclosed in single quotes, for example:

```
'ROGERS.IODF05'
```

HCD then displays the Create Work IODF Definition File panel.
8.11 Creating a new work IODF

Create work IODF file

Specify the volume you wish the IODF to be allocated to (prompting is available for this field) and press F4 for a list of volumes. If your IODF is SMS-managed, the volume serial number is not required.

Specify whether you want activity logging to be enabled. If you opt for this function, then the activity log is displayed as the last panel when exiting following an update to the IODF.

At this time you must also specify the size of the new work IODF in terms of the number of 4 KB blocks.
8.12 Defining configuration data

The next panel displayed after choosing Option 1 from the primary option menu is shown in Figure 8-12. This panel displays all the objects for which the HCD dialog provides action lists, where you can define the characteristics and the relation between these objects.

From this panel, you can go step-by-step to define, change, or delete the following:

- Operating system configurations
- EDTs
- Generics
- Esoteric groups
- Servers
- Logical Channel Subsystem
- Logical Partitions (LPs)
- Channel paths
- Control units
- Devices
- Consoles

Before using the dialog of HCD to define a hardware configuration, you should have a plan of what your configuration should look like, and what you have to do to accomplish that. Preferably, the requirements of your configuration should be established in a configuration plan. See z/OS Hardware Configuration Definition Planning, GA22-7525, for a z/OS configuration.
8.13 Operating system definition

Operating system definition

It is recommended to define the operating system configuration before you define anything else. An operating system (OS) configuration defines the data that is used by z/OS at IPL, to build its I/O control blocks. An IODF can contain more than one z/OS configuration. At IPL time, one of them is selected.

Figure 8-13 shows the elements (I/O devices, EDTs, Esoteric, and NIP consoles) and their relationship within the z/OS operating system. A NIP console is the console used by the nucleus initialization program (NIP), a z/OS program running at IPL time. NIP is in charge of initializing all z/OS components and subsystems during the IPL procedure. At early stages of this initialization, there are no multiple-console support (MCS) consoles available for communication with the operator. The NIP console is then used for such communication. The first message, Specify System Parameter is displayed on this device.

At IPL, the z/OS PARMLIB members identify the IODF and the operating system configuration (you may have more than one per IODF) and the EDT identifier.
8.14 Defining an operating system

You define the operating system from the HCD primary menu shown in 8.10, “HCD primary menu” on page 509. From this menu, select Define, modify, or view configuration data. When the next panel appears, select Operating system configurations.

Figure 8-14 shows two configurations already defined (L06RMVS1 and MVSW1). To add a new configuration, use F11.

Add Operating System Configuration panel

Complete the Add Operating System Configuration panel with the name of the operating system you would like to define. As shown in Figure 8-14, we have defined MVSNEW.

After you define the operating system, you can define the EDTs.
8.15 EDT and esoterics

Eligible device table (EDT)
An eligible device table (EDT) has several lists of devices, where each list is associated with an esoteric name. It is used at allocation time to generate a list of the devices that are candidates to receive a new data set. A z/OS operating system may have more than one EDT, but only one can be active at any one time. In Figure 8-15, the EDT would have an esoteric list of devices for SYSDA, PRODDSK, and TESTDSK.

Understanding I/O device allocation in z/OS
When you submit a job, you should identify I/O devices required by the data sets required by the job. The device information can be obtained from either a catalog (if the data set already exists), SMS overrides, or specific UNIT parameters on DD statements. Before the job (and its steps) can start execution, a z/OS initiator component must allocate all those devices to the data sets required by the job. “Allocation” in this case means to establish a link between the logical definition of the data set and the physical device where it is, or will be, located. It can also mean to make a logical connection between a requiring task (such as a batch step task) and the data set already located in a device. The allocation can also be done after the start of the step execution by its code invoking the z/OS function of DYNALLOC.

There are three ways to specify a device where the data set will be allocated for a job using the UNIT parameter on a DD card, by using:

- A specific device number
- A generic device type
An esoteric device group described in EDT

**Indicating a specific device number**

To request a device explicitly for a data set in the job, specify a device number on the UNIT= parameter or on the corresponding dynamic allocation parameter in the DYNALLOC service. If that device is available, z/OS allocates the device to the data set in the job. However, if the device is unavailable (for example, a tape drive allocated to another job), your job may wait until the device is available for allocation or being cancelled by timeout.

**Specifying a generic device type**

z/OS logically groups device types with similar characteristics and assigns the group a generic name. Such a group is called a generic device type. z/OS, for example, groups 3390-1 and 3390-2 into a generic device type named 3390. Any time a program allocates a 3390, z/OS interprets it to mean any of the devices of that generic device type.

To request a device allocation, you can specify a generic device type on the UNIT= parameter. z/OS allocates a device from the specified generic device type. If you code the following DD statement, then z/OS allocates a device from generic device type 3390:

```
//OUTPUT DD UNIT=3390,...
```

However, generic device type 3390 should not be confused with specific four-hexabyte character device number 3390. To avoid having your specification misinterpreted as a specific device number, use the following notation for the device number:

```
//OUTPUT DD UNIT=/3390,...
```

**Specifying an esoteric device group described in EDT**

A job that specifies an esoteric device group is requesting z/OS to allocate any device from that group. An esoteric device group can include devices of different generic device types. In Figure 8-15 on page 514, we have PRDDSK and TESTDSK.

The device preference list indicates the preferred sequence of device type candidates to which to allocate the data set. It is used when the esoteric definition has more than one device type. Devices belong to one of the following classes:

- Channel-to-channel adapters
- Communication devices
- Direct access devices
- Display devices
- Character readers
- Tape devices
- Unit record devices

All of the devices that you assign to an esoteric device group must be of the same device class with the following exception: you can define esoteric device groups that contain devices from both DASD and tape device classes. To request device allocation, you can specify an esoteric device group name on the UNIT= parameter on the DD JCL statement.

As shown in Figure 8-15 on page 514, SYSDA is the esoteric group name for three 3380s (device numbers 0181, 0182, and 0183) and four 3390s (device numbers 0190 through 0193). When UNIT=SYSDA appears on a DD statement, units 0181, 0182, 0183, 0190, 0191, 0192, and 0193 are eligible candidate devices.

TESTDSK is the esoteric group name for two 3390 DASDs (device numbers 0191 and 0192). PRODDS is the esoteric group name for three 3380 DASDs (device numbers 0181, 0182, and 0183).
8.16 How to define an EDT (1)

Defining an EDT
Before you can define EDTs, you must have defined an operating system. You define an EDT as follows:

▶ On the primary task selection panel, select Define, modify, or view configuration data. On the resulting panel, select Operating system configurations. HCD displays the operating system configuration list of all operating system configurations currently defined in the IODF.

▶ On the Operating System Configuration List panel, select the z/OS configuration, shown in Figure 8-16 as MVSNEW. The next panel displayed allows you to select to work with EDTs, as shown on “How to define an EDT (2)” on page 517.

▶ Select an operating system from the OS Configuration List panel, using the / action key and then select Work with EDTs Option 5 from the context menu.
8.17 How to define an EDT (2)

Defining an EDT for a new configuration

For a z/OS operating system, you have to define at least one eligible device table (EDT). An EDT can consist of one or more esoteric device groups and names of the generic device types. Esoteric device groups are installation-defined groupings of I/O devices.

A z/OS configuration can contain more than one EDT; z/OS is told which one to use at IPL time. If there are no EDTs defined in the IODF, as shown in 8.17, “How to define an EDT (2)” on page 517, the EDT list is empty. Use F11=Add to add a new EDT. The data-entry fields are shown in Figure 8-18 on page 518 with sample data.

For background information about I/O device allocation in z/OS that you need to understand before defining EDTs and esoteric groups, refer to z/OS Hardware Configuration Definition Planning, GA22-7525.
8.18 Defining an EDT identifier

An eligible device table (EDT) is an installation-defined and named representation of the I/O devices that are eligible for allocation. Using HCD, you define EDT information in an IODF. At IPL or dynamic configuration, information in the IODF and UIMs is used to select the EDT.

Add EDT panel
You specify an EDT identifier, 0A (any two alphanumeric characters) and choose a name (MVSNEW) as the configuration ID, as shown at the top of Figure 8-18. After you press the Enter key, HCD displays the EDT List panel.

EDT List panel
The EDT List panel now shows the new EDT identifier, as illustrated in the lower part of Figure 8-18.

LOADxx member for IPL
When you create the LOADxx member, you specify an IODF statement to identify the production IODF to be used. Other PARMLIB members identify the operating system configuration that is used to IPL the system. The IODF identified becomes the active IODF for the system.
8.19 How to add an esoteric

When defining eligible device tables (EDTs) for I/O devices:

- Decide how many EDTs to define.
- Specify a 2-digit identifier for each EDT, for example, 20.
- Decide how many esoteric device groups to define in each EDT.
- Specify a name for each esoteric group.

**Esoteric device group**

An esoteric device group identifies the I/O devices that are included in that group. The name you assign to an esoteric device group is called the esoteric name. To request allocation of a device from an esoteric device group, specify the esoteric name on the UNIT parameter of a JCL DD statement. The name esoteric device group is often shortened to esoteric group or simply esoteric.

**Adding an esoteric**

On the primary task selection panel, select **Define, modify, or view configuration data**. On the resulting panel, select **Operating system configurations**. HCD displays the operating system configuration list of all operating system configurations currently defined in the IODF.

On the Operating System Configuration List panel, select the z/OS configuration you are working with. The next panel displayed allows you to select to work with EDTs by selecting **Option 5, Work with EDTs** and from the next panel select the EDT identifier using the / action
key. Then, from the next panel, select Option 4 **Work with esoterics** from the context menu as shown in Figure 8-19 on page 519, and press Enter.

To define an esoteric group in an EDT with device numbers, refer to 8.51, “Assigning a device to an esoteric” on page 561.
8.20 Adding an esoteric

Add an esoteric
The Esoteric List panel appears, as shown on the top of Figure 8-20. Press F11 and the Add Esoteric panel appears and then you can add an esoteric name. Figure 8-20 shows the esoteric name SYSDA has been added.

Esoteric token and catalogs
The esoteric token is an optional value. In the past there have been access problems with data sets cataloged with an esoteric device group name. HCD arranges esoterics alphabetically, but the z/OS catalog contains the EDT index entry pointing to the esoteric. After HCD has reordered the esoterics, allocation searches the incorrect device for a data set. If you specify an esoteric token, this token will be used as the entry point to the catalog. Specify the token such that your existing esoteric or non-alphabetic order is maintained.

You should be aware that deleting an esoteric name may cause catalog problems because the catalog keeps the esoteric information used to allocate the data set at creation.

Virtual I/O (VIO)
Virtual I/O (VIO) refers to data set allocations that exist in paging storage only. z/OS does not use a real device unless it must page out the data set to a paging device. Programs that use VIO data sets access them just as if the data sets were allocated to a real I/O device. VIO is usually only set on for the user-defined esoteric called VIO.
8.21 Defining switches

Switches (also called directors) are kinds of devices with multiple bidirectional ports to which channels and control units may be attached. There are two types of switches: ESCON and FICON. To define switches and their associated ports in HCD, you need to:

- Define switch characteristics
- Define connections to CHPIDs, CUs, and other switches
- Define switch configuration data (port matrix)

On the primary task selection panel, select Define, modify, or view configuration data. On the resulting panel, select Switches. HCD displays the list of all switches currently defined in the IODF, as shown in Figure 8-21.

If there are more than one switch control unit and device, the list entry gets an indication (>, which is not shown here).

With the F20=Right key, you can scroll to the right part of the Switch List panel. Up to nine switch control units and devices can be shown. If there are more, an indication is given for the corresponding entry (Yes in column More? on the right part of the Switch List panel). These additional switch control units and devices can be viewed, for example, on the Port List for port FE.
Types of switches
Using HCD, you can define both ESCON switches (ESCON Directors, such as 9032-5) and a FICON FC switch (type 2032) or a generic FC switch (type FCS). The FICON FC switch supports an internal switch control unit and device (listed in the right side of the panel), whereas the generic FCS switch does not.

To define switches in your configuration, you need to know the switch type (for example, 9032-3 or 9032-5). You need to decide on:

- Switch ID, such as 01.
- Switch control unit number and switch device number (also called CUP).

You must define a switch control unit and device to allow software to communicate with the switch to update the switch configuration and to validate the switch data. When you specify a control unit number and device number in the HCD Add Switch panel, a control unit and device are defined to be the same type as the switch.

Switch connections
You also use HCD to define the connections of the switch ports to channels or control units, and also to define a matrix of all possible connections between any two switch ports.

The matrix information can be migrated into HCD from an ISPF table containing a switch configuration from an active ESCON Director (ESCD), a FICON Director, or from a saved ESCD file. Use the Migrate Switch Configuration Data option on the HCD Migrate Configuration Data panel.

Obviously, this matrix can also be changed through the switch console.
8.22 Adding switches

Adding a switch
Press F11 on the panel shown in Figure 8-21 on page 522 to add a switch. This displays the panel shown in Figure 8-22.

When you fill in the fields on this panel, the Add Switch function results in HCD optionally generating the switch's control unit and I/O device definitions used for host communication. This CUP device can later be defined to the appropriate operating system configurations.

On this panel, HCD allows you to define the switch itself, the range of ports installed on your director, the switch device number, and the switch control unit. If the switch control units already exist, they are automatically connected to a control unit port on the newly defined switch. This ensures that the definitions of switch control unit and switch device are consistent. Likewise, when deleting a switch, the switch control unit and switch device are deleted as well. However, you still have to perform the following HCD steps:

1. Connect the switch control unit to the server (this also establishes the switch device-server connection).
2. Connect the switch device to the operating system.

After you have entered the new switch definition data, all defined switches are displayed in the HCD Switch List panel.
8.23  Defining servers

When an HCD panel refers to processor, this means server.

You can define more than one server in an IODF, and for each defined server you can configure server-related data at POR. On the primary task selection panel, select Define, modify, or view configuration data, shown in Figure 8-10 on page 509. On the resulting panel, select Processors (Option 3 shown in Figure 8-12 on page 511). HCD displays the Processor List of all operating system configurations currently defined in the IODF, as shown in Figure 8-23.

Adding a new server

Use F11=Add to add a new server to the Processor List, as shown in Figure 8-25 on page 528.

A server can run in either of two modes:

- Basic mode
- LPAR (partitioned) mode

Basic mode

In Basic mode, you run only one operating system image on the server and it controls all the resources (memory, channels, and so forth) of the server. This mode is no longer available for z990 and z9 EC model servers.
**LPAR mode**

The Processor Resource/Systems Manager™ (PR/SM) feature contains a function called logical partition (LPAR), which allows a single server to run multiple isolated and independent operating system images (including the Coupling Facility control code (CFCC), the operating system of a Coupling Facility) in logically partitioned (LPAR) mode.

Each operating system has its own logical partition, which is a separate set of system resources including:

- A portion of central storage (also called real storage)
- One or more CPUs. The CPU can be dedicated or shared to the logical partition (LP). When the CPUs are shared, you assign a weight to guarantee the use for each partition of a certain CPU share or to impose a limitation (capping).
- Channels, which can be shared (also called multiple image facility (MIF)), reconfigurable or dedicated.
8.24 Information for defining a server

Figure 8-24 describes the information required to define a new server, as shown in Figure 8-25 on page 528. To define servers in your configuration, you need to specify:

- **Server ID** - an 8-byte alphanumeric name that you assign to identify the server in HCD (such as SCZP100).
- **Support Level** - depending on the server type/model, there may be more than one support level for the server type. If the server has several support levels, HCD displays another panel showing a list of available support levels for the server. The support level defines the supported channel path types, and the features such as dynamic I/O reconfiguration, EMIF, and Coupling Facility support, depending on the microcode level.
- **Server Type and Model** - such as 2084 B16.
- **Configuration Mode**: Basic or LPAR - if a server is in LPAR mode, you must define partitions in your configuration.
- **Number of channel subsystems**, as you planned.
- **Serial number** - if you specify a serial number, the system uses the number to verify that it is updating the correct server during IOCDS download.
- **Description** - text that you use to describe the server.
- **SNA address (network name) and CPC name** - for a server located in an S/390 microprocessor cluster (that is, several servers connected in a token-ring hardware management console LAN), you need the system network architecture (SNA) address of its support element in that LAN and its central processor complex name, usually the same as the Server ID.
8.25 Defining a server

Figure 8-25  Add Processor panel

How to define a server

Figure 8-25 shows the panel used to define a server. To display this panel, select Define, modify, or view configuration data, shown in Figure 8-10 on page 509. On the resulting panel, select Processors (Option 3 shown in Figure 8-12 on page 511). This displays the list of the servers that have been defined. From the Processor List panel, press F11 to add a server. The data entry fields are shown with sample data in Figure 8-25.

On the Add Processor panel, you can specify the network name and the CPC name, when the server is configured in an S/390 microprocessor cluster. Use Prompt on the Add Processor panel for the SNA addresses for those servers that are currently configured in the S/390 microprocessor cluster.

Number of LCSSs

As shown in Figure 8-25, the number of channel subsystems defined is two (LCSS0 and LCSS1). See Figure 8-27 on page 530.
8.26 Working with LCSSs

The top portion of Figure 8-26 shows the newly added server, SCZP100, on the Processor List panel.

From the Processor List panel shown in the top portion of the visual, placing a forward slash (/) next to the newly defined server displays the “Actions on selected servers” pop-up window (shown in the bottom portion of Figure 8-26). From this panel, select Option 12 (Work with channel subsystem).

From the HCD primary option menu you can define partitions as follows:

- On the primary task selection panel (shown in “Defining configuration data” on page 511), select Define, modify, or view configuration data. On the resulting panel, select Processors (Option 3). HCD displays the Processor List of servers currently defined in the IODF, as shown in the top portion of Figure 8-26.

- On the Processor List panel, select the server and the Work with channel subsystems action from the context menu. HCD displays the Channel subsystem list panel showing the currently defined partitions for the designated server, as shown in Figure 8-27 on page 530.
8.27 Logical channel subsystems defined

Defining the channel subsystems
The channel subsystems were defined on the Add Processor panel; see Figure 8-25 on page 528. The two channel subsystems defined are shown in Figure 8-27 on the Channel Subsystem List panel.

Logical partitions defined to an LCSS
To assign partitions (LPs) to an LCSS, either type p or type a / next to the CSS ID.
8.28 Adding a logical partition (LP)

Figure 8-28  Add Partition panel

Defining logical partitions
For the z990 and z9 EC servers, logical partitions must be associated with an LCSS. The z9 EC supports up to 60 logical partitions.

To define partitions, you need to decide on:

- A unique LP name (such as MVSPROD), anyone you choose defined in HCD
- A logical partition ID to be defined in HMC
- A MIF ID to be defined in HCD in the option partition number
- Whether the LP is to be used by an operating system, such as z/OS (OS option), or by a CFCC partition (CF option)
8.29 z9 EC LPAR server configuration

Figure 8-29 shows an example of an LPAR server configuration, where:

1. Logical partition names are specified in the I/O definition process (HCD or IOCP) and must be unique for the logical partition across all LCSSs in the z9 EC.
2. Logical partition MIF ID is specified in the I/O definition process (HCD or IOCP) in the partition number field and must be unique x'00' to '0F' for all the logical partitions across each LCSS in the z9 EC.
3. Logical partition ID is specified (by the user) in the z9 EC Image profile (for the LP) in HMC and must be unique x'00-3F' for the LPs across all LCSSs in the z9 EC.

Channel spanning
The z9 EC LCSS is extended to provide the concept of spanning. This is an extension to the MIF shared channel concept. Spanning channels provides the ability for the channel to be configured to multiple LCSSs. Consequently, they may be transparently shared by all or any of the configured LPs in all the LCSSs. This can be done regardless of the channel subsystem to which the LP is defined. Refer to Figure 8-29, where the CHPID 04 is spanned. Since it is not an external channel link, there is no PCHID assigned. CHPID 06 is an external spanned channel and has a PCHID assigned. A channel is spanned if the same CHPID number is assigned to the same PCHID in multiple LCSSs. In the case of internal channels (IC links, HiperSockets), the same applies, but there is no PCHID association. The following channel types can be spanned: HiperSockets, IC, FICON Express, FICON Express2, OSA Express, OSA Express2, ICB4, ICB3 and ISC3.
8.30 Channel operation mode

In relation with the use of a channel by a logical partition (LP), a channel can be:

- Dedicated
- Reconfigurable
- Shared or multiple image facility (MIF)

Channel operation modes
According to how operational modes are related with the LPs, a channel may be:

- Dedicated - If you want only one LP to access a channel path, specify that channel as dedicated. You cannot reconfigure a dedicated channel path. This is the default mode. All channels can be dedicated.

- Reconfigurable - If you want only one LP at a time to access a channel path and you want to be able to manually reconfigure the channel from one LP to another, specify that channel path as reconfigurable. All channel types can be reconfigurable. When defining a reconfigurable channel, you decide which LP will be assigned to it at LP activation time and the LPs that can be given access to the channel later on. This is indicated through a candidate list. Refer to “Defining an access and a candidate list” on page 541.

- Shared or MIF (multi-image facility) - If you want more than one LP to access a channel simultaneously, specify that channel path as shared. Only the FCP channel cannot be shared. With MIF, LPs can share the same device through a single shared physical channel or set of shared channels. The major advantage of MIF is that it reduces the number of channels, which allows a cheaper and simpler I/O configuration. MIF enables resource sharing across LPs within a single LCSS or across multiple LCSSs. When a channel is shared across LPs in multiple LCSSs, this is known as “spanning”. When defining the shared channel, you decide which LPs will be assigned to the channel. This is indicated through Access and Candidate lists. The default with HCD for shared channels is to have all the partitions in the Access list so the channel is online to all LPs on the server.
8.31 Channel types

Channel types
Following is a short description of the channel types.

ESCON channels
An ESCON channel can be connected to an I/O controller (CNC) or can be connected to other ESCON channel or a FICON convertor (FCV) - named channel-to-channel (CTC). CTCs allow the flow of data between central storages of different LPs in the same or in distinct servers. In each logical partition that can communicate with another server complex through a shared ESCON CTC channel path, you must specify the logical address of the ESCON CTC control unit. You can specify MIF IDs when defining LPs, and you can specify these MIF IDs as the logical address for a CTC control unit.

FICON channels
FICON channels are based on the fibre architecture, with some differences to implement more security. FICON channels increase the server's I/O capacity; each FICON channel has, on average, performance that is equivalent to four ESCON channels. The FICON channel (FC) requires either a FICON interface at the controller, or it needs to be connected to a fiber channel switch (director) port.

FICON bridge channel (FCV)
Using the FICON bridge feature on the 9032-005 ESCON Director, you can attach ESCON controllers to a FICON channel. The FCVs offer a migration path for ESCON CNC channels.

Figure 8-31 Channel types
An FCV channel path occupies eight port addresses on the switch. To model the FCV bridge within HCD, consider the following: whenever you connect an eligible port address to an FCV channel path, you must set all other port addresses occupied by the FCV bridge to "uninstalled".

**Fibre channel protocol (FCP) channels**

Fibre channel protocol (FCP) channels are a real implementation of the fibre channel architecture, used in z9 EC by Linux LPs that are running (or not) under z/VM.

**Coupling Facility (CF) channels (ICP, CBP, and CFP)**

CF channels (also called *CF links*) connect z/OS with CFs, and connect CFs with CFs. The types are as follows:

- **ICP**
  This is a microcode-defined link to connect a CF to a z/OS LP in the same z9 EC server. The CF link itself is known as an internal link (IC), requiring two CHPIDs to be defined. The link rate is greater than 2 GB/sec.

- **CBP**
  This is a copper link available to connect IBM System z servers; that is, the z/OS LPs and the CF LPs are in different servers. The CF link itself is known as an integrated cluster bus (ICB4), requiring a maximum distance between the two servers of seven meters. The link rate is 20 GB/sec. They are directly connected to the MBA in the server cage through a dedicated STI.

- **CFP**
  This is a fiber link defined in peer mode available to connect IBM System z servers; that is, CF LPs and z/OS LPs are in different servers with a distance greater than seven meters. The CF link itself is known as intersystem channels (ISC3), requiring a maximum distance between the two servers of 10 km with unrepeated channels, or 20 km with an RPQ, or 40 km with Dense Wave Division Multiplexing (DWDM) connections. The link speed is 200 MB/sec (for distances up to 10 km), and 100 MB/sec for greater distances (RPQ).

**Open systems adapters express (OSA-Express2)**

OSA-Express2 comprises integrated network controllers packed together with I/O channels located in the z9 EC I/O cages. Depending on the features installed in an OSA-Express2, we may have access to different types of networks, such as 10 GbE LR, 1000BASE-T Ethernet, and others. OSA-Express2 in z9 EC works as four different types of channels:

- OSD (queued direct I/O - QDIO)
- OSE (non-queued direct I/O)
- OSC (OSA-express integrated console controller)
- OSN (open system adapter for NCP)

**HiperSockets or internal QDIO (IQD)**

This is an LIC providing a logical channel with a very fast TCP/IP communication path between LPs in the same server. The operating systems exploiting such a facility can be z/OS, z/VM, VSE/ESA, and Linux. HiperSockets provide internal "virtual" LANs that act like TCP/IP networks in IBM System z servers.
8.32 Information required to add channels

- CHPID and PCHID
- Type of channel
- Operation mode
  - Dedicated
  - Reconfigurable
  - Shared
- DCM-managed
- Access and candidate Lists
- Dynamic entry switch ID
- Entry switch ID
- Entry port

Figure 8-32 Information required when defining channels to HCD

Information required to add channels

To define channel paths in your configuration, you first make your physical layout decisions such as your CHPID numbers for channels, number of channel paths, channel path type, and switch directors (ESCON and FICON) information, if channel paths are connected to a switch. The switch information includes entry switch ID, ports, and dynamic switch ID.

Then you need to decide on an operation mode (dedicated, reconfigurable, or shared) and the associated access and candidate lists.

Note: The icons shown in Figure 8-32 are for the instances and defined channel types displayed on the Hardware Management Console.
8.33 Working with channel paths

We may say that channel and channel path are interchangeable terms with the same meaning. Usually HCD manuals tend to use channel path instead of channels.

On the primary task selection panel, select **Define, modify, or view configuration data**. On the resulting panel, select **Processors**. HCD displays the Processor List of servers currently defined in the IODF.

On the Processor List panel, select the server you are working with (in this example, a z990) by typing action code s.

HCD displays the Channel Subsystem List panel, in the bottom portion of Figure 8-33, showing the two LCSSs (0 and 1) defined for the selected server.

**Select an LCSS to add the channel**

On the Channel Subsystem List panel, type action code s to display the Channel Path List panel. Initially, this panel should be empty; after channel paths are defined, you see them in the list.
8.34 Adding channel paths dynamically

The Channel Path List panel eventually shows all the channel paths defined for the selected server for each logical channel subsystem.

To add a channel path, use F11=Add; this displays the Add Channel Path panel shown in Figure 8-35 on page 539.

You can now add, modify, and delete channel path definitions. However, if you define a device as static or installation-static, you cannot delete or modify the channel path definition to that device. You may also be able to dynamically add, delete, or modify CF channels (links).

To dynamically add a channel path, use the following general steps:

- Describe the new I/O configuration definition in an I/O definition file (IODF) that you create through HCD, including the new channel.
- Activate the new IODF through the Activate z/OS command or through an HCD panel.
- Install the new I/O components that you have dynamically added.

---

**Figure 8-34 Channel Path List panel**

```
CBDPCHF6 Channel Path List Row 1 of 5 More:
Command ===> Scroll ===> CSR

Select one or more channel paths, then press Enter. To add use F11.

Processor ID . . . . : SCZP100
Configuration mode : LPAR
Channel Subsystem ID : 0

DynEntry Entry +
/ CHPID Type* Mode* Switch * Sw Port Con Mngd Description
  02 FC SHR 01 01 D8 No
  03 FC SHR 02 02 D8 No
  04 FC SHR 01 01 E8 No
  05 FC SHR 02 02 E8 No
  80 CNC SHR ___ ___ ___ No

************************************************************************ Bottom of data **************************************************************************
```
### 8.35 Adding a channel path

#### How to add a channel path

Fill in the fields on the Add Channel Path panel. Because the server ID matches a z9 EC previously defined, you can operate only in LPAR mode.

This panel is for LCSS 0. Next you define the channel path characteristics by defining the following:

- Channel path ID (CHPID), shown as 00
- PCHID
- Number of CHPIDs, shown as 1
- Channel path type, shown as FICON native in “Channel types” on page 534
- Operation mode, shown as SHR
- Managed, shown as YES
- Description
- Dynamic entry switch ID, shown as 01
- Entry switch ID, shown as 01
- Entry port, shown as D1

**Note:** On a z9 EC server, you need to specify the physical channel identifier (PCHID) associated with the channel path identifier (CHPID).

However, with the CHPID mapping tool (CMT) you can map CHPID and PCHIDs more easily. It means that you are not forced to declare the PCHID now. Work without the PCHID and after finishing the configuration, submit the IOCP (generated by HCD) to the CMT for PCHID assignment.

- Number of CHPIDs, shown as 1
- Channel path type, shown as FICON native in “Channel types” on page 534
- Operation mode, shown as SHR (shared)
Managed, shown as YES

**Note:** You can define an ESCON channel path as being managed by dynamic channel path management (DCM), an IRD function. DCM dynamically switches the channel connection among control units in order to optimize I/O activity and balance channel utilization.

In order to implement DCM, a managed ESCON channel path must connect to an ESCON dynamic switch, and may be used for control units that connect to the same switch. If a channel path is defined as managed, it must be defined as shared and must specify an I/O cluster name.

An I/O cluster is the sysplex that owns the managed channel path. All systems of the sysplex on the given server are allowed to share the managed channel path. A managed channel path cannot be declared as connected to a control unit in HCD.

DCM is not available for FICON channels.

HCD distinguishes between the dynamic and entry switch identification when defining the connections of a channel path. The dynamic switch is the switch holding the dynamic connection; the entry switch is the switch to which the channel path is physically plugged. These IDs can be different when the switches are connected in a cascade of two.

**Note:** You can define multiple channels in one step. If you do so, and have also specified an entry switch and entry port for the channel path, HCD displays another panel where you can specify the entry switch and port number for the subsequent channel paths.

It is not mandatory to declare the Entry Port ID (D1 in the panel), describing the port ID in the switch where the channel is connected. Each channel, during the initialization process, can discover the switch port ID that it is connected to. This allows a double-check of the definition.

After you press Enter on the Add Channel Path panel, HCD displays the Define Access List panel.
8.36 Defining an access and a candidate list

Channel path access list
An LP that is on a shared channel's access list can access the channel when the LP is initially activated. When a channel is dedicated, you specify one LP on the channel path access list. When a channel is shared, you can specify more than one LP on the channel access list. Figure 8-36, in the top portion, shows the access list with LP MVSPROD only.

Channel path candidate list
If you do not include all LPs in the access list, you are prompted for the candidate list (for reconfigurable and shared channel paths) after pressing Enter. This panel is shown in the lower portion of Figure 8-36, and a second partition, TEST, is then selected.

An LP that is on a channel's candidate list can eventually access the channel. An LP on this list can access the channel when the channel is manually configured online (through a z/OS Config command) to the LP.

In this example, we added CHPID 80 to the access list of MVSPROD and the candidate list of TEST. Use the / key to select the LPs.
8.37 Adding a control unit

Control units
An I/O control unit (or controller) provides the logical capabilities necessary to operate and control a set of I/O devices, and adapts the characteristics of each device so that it can respond to the standard form of control provided by the channel subsystem. Refer to Chapter 9, “DS8000 series concepts” on page 577 for more information on DASD I/O controllers.

Communication between the control unit and the channel subsystem takes place over a channel path. The control unit accepts control signals from the channel, controls the timing of data transfer over the channel path, and provides indications concerning the status of the device.

The I/O device attached to the control unit may be designed to perform only certain limited operations, or it may perform many different operations. A typical operation is moving a recording medium and recording data. To accomplish its operations, the device needs detailed signal sequences peculiar to its type of device. The control unit decodes the commands received from the channel subsystem, interprets them for the particular type of device, and provides the signal sequence required for the performance of the operation.

A control unit may be housed separately, or it may be physically and logically integrated with the I/O device (DS8000), or the channel subsystem (OSA-Express2). For specific reasons in certain cases, the same physical control unit must act as several control units called logical control units, and in HCD you should define each logical control unit.
How to add a control unit

Here the expression control unit really means logical control unit. The configuration being pictured in Figure 8-37 on page 542 is a 2105 (Shark) with two logical control units (1700 and 1701) using MIF channels attached to two ESCON directors. When defining control units, you need to define the following:

- Control unit characteristics
- How the control unit is attached to a server

The steps for the configuration in Figure 8-37 on page 542 are as follows:

1. Define the ESCON channels, sharing them between TEST and MVSPROD LPs and connecting them to the ESCD switches. In this example, shown in Figure 8-36 on page 541, the channels have already been defined.

2. Define the two 2105's control units, specifying the CHPIDs.

3. Define the DASD devices and connect them to the logical control units.
8.38 Information required to define a control unit

Figure 8-38  List of items to define control units

Information required to define a control unit
Here we discuss the items needed to correctly define a control unit to HCD:

- Control unit type and model, such as 9343-1
- Control unit serial number
- Description - text that you use to describe the control unit
- Servers to which the control unit connects
- Control unit number, such as 1700 (just for HCD correlation)
- Connections to switches
- Channel path IDs (CHPIDs) where the control unit can connect
- Link address, the switch port number where the control unit is connected with the respective channel in the switch. This information is mandatory, because the channel must know this port ID in order to send an ESCON or FICON frame to the control unit.
- Information to attach the control unit to the channel path of the server:
  - Device unit address ranges that the control unit recognizes
  - I/O concurrency level - classification of a control unit based on its ability to concurrently operate and control the activity of attached devices without causing loss of control or data
  - Logical address - known as the CUADD value
Adding a control unit

On the primary task selection panel, select **Define, modify, or view configuration data**. On the resulting panel, select **Control units**. HCD displays the Control Unit List panel showing all control units currently defined in the IODF (on the first definition, this panel is empty).

Column #CSS shows the number of channel subsystems to which a control unit is connected. The column contains a value only if the connection exists. Here the number five indicates five LCSSs for sure located in different servers.

Column #MC shows the number of DCM managed channels defined for the connected servers. 

Observe that in Figure 8-39, we are not defining switches (2032 and 9032), we are just defining the switches as control units (and later with CUP devices) so that the software can set the connections of the ports.

Adding additional control units

Press F11 to add the second control unit. After filling in the details, press Enter to display the Add Control Unit panel. Pressing Enter again returns you to the Control Unit List panel.

The following prerequisites must be met for this function:

- The control unit must support ESCON or FICON attachments and not be used for channel-to-channel (CTC) connections.
The control unit must have physical switch/port connections (switch entry ports) defined.

Channel paths that use the connected switch as a dynamic switch must exist.

HCD then automatically selects the channel paths and link addresses according to the following rules for each server that has been defined:

- All channel paths that use a switch that connects to the control unit as a dynamic switch are candidates for assignment.
- The channel paths are sorted ascending by the value of the reversed channel path ID. The resulting sequence is again sorted ascending by the number of connected devices.
- The connected control unit ports are ordered ascending by the numbers of already connected control units and path connections, respectively.
- For each connected switch port in the resulting sequence, the channel paths are tested in sequence. If the switch port can be used as a link address, the CHPID/link address is taken.
- A maximum number (up to 8) of possible CHPID/link address combinations is assigned.
8.40 Defining a 2105 control unit

Figure 8-40 is showing the definition of the Shark 2105 control unit:

1. Select Control Units from the Define, Modify or View Configuration Data panel.
2. Press F11 to add a new control unit.
3. Fill in the fields on the panel as follows:

   The Add Control Unit panel can also be used to specify the switches and ports the control unit is attached to. In the example, this logical control unit is connected, respectively, to switch 01 through port IDs 40 and 50; and to switch 02 through port IDs 40 and 50.

   If you specify Yes for “Define more than eight ports,” a further panel is displayed to allow you to specify up to 32 control unit switch/port connections.

   If you specify Yes for “Propose CHPID/link addresses and unit addresses,” HCD suggests control unit-to-server attachment parameters (channel path/link addresses and the unit address range) based on the switch/port the control unit is connected to. HCD proposes up to eight channel path/link address pairs, starting with the channel path that has the lowest number of devices attached to it.

4. After typing in the details, press Enter to display the Select server/Control Unit panel, shown in Figure 8-41 on page 548.
8.41 Selecting a processor/control unit

Select processor for control unit

Up to now, you have defined the control unit and established its links with switch ports. Still missing are the channels and their servers that access such control units.

After pressing Enter on the Add Control Unit panel, HCD displays a list panel, shown in Figure 8-41, that shows all the defined servers. Select the server you wish to attach the control unit to by typing a slash (/), pressing enter, and then selecting the Select (connect, change) action from the context menu, as shown in Figure 8-42 on page 549.
8.42 Servers and channels for connecting control units

Attaching servers and their channels to control units

When a control unit is attached to multiple servers, you can use the group connect action from the context menu (or action code g). This group action is particularly useful when performing symmetric changes, for example, on servers defined in an HMCplex. The changes are applied to all selected servers, when you issue the change action against a group of servers.

When you issue a change or group connect action, the panel for dependent control unit information is displayed, as shown in Figure 8-43 on page 550.
8.43 Defining server attachment data

The Add Control Unit panel specifies channels that connect the control unit to the server and also the number of devices served by the control unit. If the control unit is attached to a switch, you have to define a link address for each channel path. The link address is the port to which the control unit attaches. If the control unit attaches only to one port, the link address is the same for each channel. For addressing the target control unit in a fabric FICON containing cascade switching (two connected switches), a 2-byte link address is used, which specifies the switch address and the port address to which the control unit is attached.

You must also specify the unit address and the number of units, that is, the unit address range of I/O devices that the control unit recognizes.

If the same physical control unit simulates several logical control units in the same link address, you have to specify a logical address (CUADD parameter). This CUADD is used by the channel for unique identification of the logical controller. Refer to “ESCON concepts” on page 390 to better understand the role of CUADD in ESCON and FICON channels.

The Protocol and I/O concurrency level applies to parallel channel interfaces. Such channels are not supported by z9 EC and z990 servers.

Press Enter. HCD displays the updated Select Processor/ Control Unit panel. Repeat defining processor attachment data for all processors the control unit should be attached to.

Press Enter to return to the Control Unit List panel.
8.44 Information required to define a device

Before you define a device to an operating system and to the channel subsystem (CSS), you must have defined the operating system, server, channel path, and control unit. For example:

- You cannot define the server data for the device, if the device is not attached to a control unit, or the control unit is not attached to a processor.
- You cannot define the EDT/esoteric group data for the device until you have defined an EDT for the operating system.

To define I/O devices in your configuration, you need to define:

- Device type and model, such as 3390-6
- Device number and unit address you want assigned to the device
- Numbers of the control units to which the device attaches (devices only connect to one logical control unit)
- Device parameters and features connected to the operating system, including whether the device supports dynamic configuration or whether a tape device is automatically switchable
- For esoteric device groups (that you named in the EDT as part of defining operating system data), which I/O devices you include in each group
- I/O devices that you allow z/OS to use as NIP consoles
SAP selecting algorithms

When SAP is selecting which channel should be tested first in order to start the I/O operation towards the device, there are two algorithms:

- Rotate (the default), to distribute the device load by all channels reaching the device
- Preferred algorithm, where the same designated channel is always tried first, recommended only for 3490 tapes
8.45 z/OS device numbering

z/OS device numbering

Operating systems need I/O device identification to uniquely address the devices for communication with the installation. A device number is the number you assign to identify a device in HCD. It is used for communication between z/OS and the operator. A device number may be any hexadecimal number from X’0000’ to X’ FFFF’, which results in a limitation of 64K devices per z/OS image. In the z9 EC, there is the possibility of roughly reaching 128K devices per LP. However, z/OS only uses the second set of 64K for alias addresses in PAV. So the same device number is shared by the base PAV address and its aliases.

The channel subsystem (CSS) also needs such unique identification for performing I/O operations to a specific device. For this purpose, the one-byte device address was introduced (also called unit address), which together with the CUADD uniquely identifies the device for an I/O operation. A unit address is defined in the control unit HCD panel and is a physical address. There is a limitation of up to 256 devices per channel per logical control unit.

Both addresses are pictured in Figure 8-45. You need three steps to define an I/O device:

- Define device characteristics and control unit connection.
- Define CSS-related definitions for a device.
- Define OS-related definitions for a device (including EDT and esoteric group assignment).
8.46 Defining a device

How to define an I/O device

You can define a group of I/O devices of the same type and with consecutive device numbers by specifying the first device number and the number of devices in the group. Then HCD applies the definition to all devices in the group. On the I/O Device List panel, you can type over the values that should be different.

Because the limitation of 64K device numbers is per LP, HCD allows you to assign the same device number to more than one I/O device in the configuration; that is, device numbers alone do not uniquely identify a device in one IODF. To clearly identify devices, HCD keeps track of each occurrence of the same device number by appending an internal suffix to it.

The correlation of operating system to partition is important only when making changes to the hardware and software definitions.

The device numbers (defined in the Add Device panel) are associated with unit addresses (defined in the Add CU panel). The number of consecutive device numbers is associated with the number of unit addresses.

The devices are defined as follows:
1. Select Devices from the Define, Modify or View Configuration Data panel.
2. Press F11 to add a new device. Complete the panel by specifying the device number, number of devices, device type, and control unit numbers the devices are attached to, for the devices you are adding.
3. Press F11 to add the 3390 devices to the 3990 control unit. Complete the panel by specifying the device number, number of devices, device type, and control unit number.

4. After specifying the details, pressing Enter displays the Device/Processor Definition panel. After confirming the responses on this panel, press Enter.

**Devices supporting parallel access volume (PAV)**

PAV allows several I/O requests being executed concurrently in the same 3390 logical volume. As shown in Figure 8-46 on page 554, when you define a PAV device in HCD, you define a device type that indicates PAV capability. Base devices are defined using a base device type, for example '3390B'. Alias devices are defined using an alias device type, for example '3390A'.
8.47 Defining device CSS features (1)

**Figure 8-47   Device/Processor Definition panel**

<table>
<thead>
<tr>
<th>Preferred</th>
<th>Device Candidate List</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Device/Processor definitions**

On the Device/Processor Definition panel, you can specify the device and CSS-related parameters and features by either typing over the fields in each column, or by selecting a processor and pressing Enter. The Define Device/Processor panel shown in Figure 8-48 on page 557 is then displayed.
8.48 Defining device CSS features (II)

You can restrict LP access to an I/O device on a shared channel path by using the explicit device candidate list, to select which logical partitions can access that I/O device. On the Define Device/Processor panel, enter Yes or No in the Explicit device candidate list field to specify whether you want to restrict logical partition access to an I/O device:

YES Specifies that only your selected LPs can access this I/O device. Note that the LP must also be in the channel access or candidate list to access the device. On the Define Device to Operating Systems Configuration panel (Figure 8-49 on page 558), place a s character to the left of each selected LP name.

NO Specifies that all LPs can access this I/O device. NO is the default; all LPs are in this I/O device’s candidate list.

Time-Out specifies whether the I/O interface time-out function is to be optionally active.

STADET specifies whether the status verification facility is enable or disable.

About Preferred CHPID, refer to 8.44, “Information required to define a device” on page 551.

After confirming the responses on this panel, press Enter to display the Define Device to Operating System Configuration panel. If the last two characters of the device number are the same as the unit address (not recommended), there is no need of defining the unit address field. Remember that the device number is the device’s nickname, and the unit address plus the CUADD is the physical identification of the device.
8.49 Defining devices to the operating system

Define devices to the operating system

Here, you may selectively, per device, either allow or not allow the access to a specific operating system (LP). This capability is very important for security, for example to deny physical access to production data bases from a developing logical partition.

Select the operating system to define the devices to, and then select the Select (connect, change) action from the context menu (action code s). The next panel is displayed, shown in Figure 8-50 on page 559.

Note: You can select all the operating systems at this time; HCD takes you through the appropriate panels.
8.50 Defining operating system device parameters

Defining operating system device parameters

Figure 8-50 is displayed after you press Enter on the Define Device/Processor panel (shown in Figure 8-48 on page 557) and when you select a processor on the Define Device to Operating System Configuration panel that shows all the defined OS configurations. You can then define the data about device parameters and features that is required by the operating system configuration.

The Parameter/Feature fields vary, depending on the I/O device type and operating system type, as follows:

- A plus sign (+) in the P column indicates that you can use F4=Prompt to get a list of possible values for the parameter/feature in the same row.
- A Yes in the Req. field indicates that a value for the parameter/feature in the same row is required.

You accomplish the change by accepting the default values or by changing the value entries and pressing Enter. The default values are set by the UIM routines for the device type. For parameters, you can specify different default values via the OS_PARM_DEFAULT keyword in the HCD profile.

- After you have defined the device parameter and feature data and pressed Enter, HCD displays the Assign/Unassign Device to Esoteric panel, shown in Figure 8-51 on page 561.
Operating system features
The following device-related z/OS operating system features should be reviewed:

- The OFFLINE feature defaults to the value specified in the UIM (which may not be appropriate for tape devices and ESCON Directors). The OFFLINE value specified here means the device is not available in z/OS (UCB) at IPL, and to the channel subsystem (UCW) in relation to the associated z/OS image. However, it can be online through a simple z/OS console VARY command.

- The DYNAMIC feature defaults to YES for devices that allow dynamic reconfiguration. Refer to 8.6, “Dynamic I/O reconfiguration” on page 501.

- Check your subsystems and applications that manage their own devices to make sure they support dynamically modified device control blocks (UCBs).

The status of a device may be changed from DYNAMIC=YES to DYNAMIC=NO, and vice versa, by a software-only dynamic change to a configuration. When changing from DYNAMIC=NO to DYNAMIC=YES, this must be the only change made to the device at that time. You are allowed, however, to make changes to a device that is currently defined as DYNAMIC=YES and at the same time change it to DYNAMIC=NO.

**Note:** The DYNAMIC parameter is shown only when the appropriate device supports the dynamic I/O configuration function.

- The SHARED feature defaults to NO. Since many z/OS installations have more than one image, this parameter should be checked and changed to YES if appropriate.

Press Enter to assign the esoterics:

1. Specify YES or NO to assign or unassign devices to the esoteric. If you do not want to assign all the devices currently being defined to this esoteric, you can limit the devices being assigned by specifying a starting device number and the number of devices to be assigned.

2. Pressing Enter returns you to the I/O Device List panel.
8.51 Assigning a device to an esoteric

Assign device to an esoteric

On the Assign/Unassign Device to Esoteric panel, overwrite the values in the Assigned column to assign (Yes) or unassign (No) devices to the selected esoterics.

If you do not want to assign a complete group of devices, you can limit the range by specifying a starting number and the number of devices. If you omit the number of devices, then 1 is assumed.
8.52 Defining an NIP console

The operator can complete the initialization process at IPL from the system console or hardware management console. The system console is the NIP console defined at HCD. You should remember that at NIP time the MCS consoles are not active yet. To define a NIP console to the operating system, do the following:

1. On the primary task selection panel, select Define, modify, or view configuration data. On the resulting panel, select Operating system configurations. HCD displays the Operating System Configuration List panel showing all OS configurations currently defined in the IODF.

2. Select an OS configuration and the Work with consoles action from the context menu (or action code n). HCD displays the NIP Console List panel (depending on the type of the selected operating system).

To define consoles, proceed as follows:

1. Define a console device and connect it to the operating system.
2. Select an Operating System Configuration and the Work with consoles action from the context menu.
3. Press F11 to add an NIP console definition.
4. On the Add NIP Console panel, enter the device number of the device you want to define as a NIP console and the order number of this console. Pressing Enter again displays the panel, showing the console just defined.
8.53 Using the CHPID mapping tool

CHPID mapping tool

Before you can build a production IODF for a z9 EC or z990 servers, with a validated work IODF, you can use the CHPID mapping tool to accomplish the task of updating or inserting required PCHIDs. Input to this tool is an IOCP input data set. To get this input, now use the task **Build IOCP input data set**, as shown in Step 3 in Figure 8-53.

Build an IOCP data set

On the primary task selection panel, shown in Figure 8-10 on page 509, specify the name of a validated work IODF and select **Activate or process configuration data**. On the resulting panel, select **Build IOCP input data set**. HCD displays the Available Processors panel. On that panel, select the processor for which you want to build the IOCP input data set.

HCD displays the Build IOCP input data set panel. On this panel, you can perform these actions:

- Enter the identification information you want to be written on the first header line of the IOCP input data set in the Title1 field.
- Specify the name of the IOCP input data set. The IOCP input data set will automatically be allocated (record length 80, record format fixed block). If the data set already exists, you will be asked to confirm replacing it with the new one.
Specify whether to build the IOCP data set for standalone IOCP by specifying either YES or NO. YES is the default. The generated IOCP statements can be used as input to standalone IOCP programs.

**Using the CHPID mapping tool**

Because the input to the CHPID mapping tool must be a standalone IOCP, in the panel shown in Figure 8-54 specifies the appropriate option.

**Input to Stand-alone IOCP? Yes (Yes or No)**

Specify or revise the following values.

- **IODF name** . . . . . . . . . : 'BOKA.IODF08'
- **Processor ID** . . . . . . . : CF01
- **Title1** : ___________________________________________________________
- **Title2** : BOKA.IODF08 - 1999-07-02 15:42

**IOCP input data set**

**Input to Stand-alone IOCP? Yes (Yes or No)**

**Job statement information**

```
//WIOCP   JOB (5765),'BOKA',NOTIFY=BOKA,CLASS=A,MSGCLASS=X,REGION=5M
//JOBLIB     DD   DSN=HCDTEST.IZPIOCP.FALC,DISP=SHR
//GO.HCDPROF DD   DSN=HCDTEST.PROFILE(MIGENH51),DISP=SHR
/*/```

*Figure 8-54  Building an IOC input data set*

Proceed as follows:

- Use the CHPID mapping tool to get PCHIDs inserted or updated in the validated work IODF. As soon as all PCHIDs are correct in the validated work IODF, the production IODF can be built.

- Go back to the Activate or Process Configuration Data panel and use the task **Build IOCP input data set** to export the I/O configuration from the validated work IODF to an IOCP data set (with PCHIDs still missing or obsolete). The I/O configuration token is passed with the IOCP statements (TOK keyword). This token is used to assure that, during the process of assigning PCHID values, the content of the IODF is not changed. Download this IOCP data set to the workstation where the CMT is running.

- Use the CHPID mapping tool with the downloaded IOCP data set. The output of a successful CMT run is again an IOCP data set which contains the original I/O definitions, together with inserted and/or updated PCHID values. The original I/O configuration token is still contained in the generated statements.

- Upload the new IOCP data set to the host and use the HCD primary task **Migrate configuration data** to import the PCHIDs from the updated IOCP data set into the validated work IODF. During this task, you select:

```
migrate option ---> 3. PCHIDs
```

- If a PCHID migration has been successfully done, you can invoke the **Build Production IODF** task (again). HCD now builds a production IODF that contains all the data that is required to write the I/O configuration data set (IOCDS) via the IOCP program to the Support Element (SE) of the machine, ready to be used for the next IML.
8.54 Build a production IODF

Although HCD validates configuration data as it is entered, a complete validation may not be performed, because data may not be defined at this time. Therefore, a "post-validation" is performed at "Build Production IODF" time. This validation might issue messages you have to deal with, according to their severity. The production IODF is not created if any errors with a severity higher than "warning" are produced.

During the validation, HCD invokes the IOCP program to perform checking on the channel packaging rules. Therefore, note that the correct version of the IOCP program must be accessible.

Depending on what is defined in the configuration, the work IODF must contain a definition for at least one operating system, one server, or one switch.

- For a z/OS operating system, the IODF must contain at least one EDT and one device.
- For a server, the IODF must contain a definition for at least one channel path, one control unit, and one device. If only channel paths of type CFP (Coupling Facility peer-to-peer) are defined for a server, then control unit and device definitions can be omitted.

Production IODF on a z9 EC server

To build a production IODF, perform the following steps:

- On the primary task selection panel, select Activate or Process Configuration Data.
From the resulting panel, select **Build Production I/O Definition File**, shown in Figure 8-56 on page 567. HCD validates the configuration data in the work IODF. If the work IODF is valid, then a production IODF can successfully be built. For work IODFs containing a z9 EC server definition, the correct PCHIDs must be defined in the work IODF before you can build a production IODF. You can use the CHPID mapping tool to either insert missing PCHIDs or to update PCHIDs in a work IODF. However, inserting or updating PCHIDs into an IODF using the CHPID mapping tool is only possible with a so-called validated work IODF that you can get in one of the following ways:

- Use the task **Build validated work I/O definition file**. This task validates a work IODF for correctness and completion, and may issue messages that describe incomplete or erroneous logical definitions. Missing PCHID values are not flagged as errors. If errors occur, correct them and restart this task. After no more errors occur, the output from this task is a validated work IODF.

- If you tried to build a production IODF without being aware of one or more missing PCHIDs for z9 EC processors, but the work IODF satisfies all other validation rules, then the output from the task “Build production I/O definition file,” too, is a validated work IODF. A message will show all CHPIDs for which the required PCHIDs are missing.

**Production IODF**

If you select option 1, which is the task **work IODF at present** (shown in Figure 8-56 on page 567), then the content of the currently built production IODF is copied to the work IODF. This ensures that the work IODF contains the latest configuration tokens of the IODF, and you can continue to use the work IODF for further updates.

If you select option 2, which is the task **new production IODF**, then the content of the production IODF is not mapped into the work IODF. In that case, you should start from the newly built production IODF when performing further changes to the I/O configuration.
8.55 Define the descriptor fields

**Figure 8-56  Define descriptor fields panel**

**Descriptor fields panel**
After pressing Enter on the Build Production I/O Definition File panel, the panel shown in Figure 8-57 on page 568 appears.

Here you specify the descriptor field 1, 2, or leave the default values. The descriptor fields describe the IODF and will be part of the HSA token. If you specify asterisks (**) for the IODF suffix in LOADxx, then z/OS uses the descriptor fields to find the current IODF. For further details about this topic, refer to *z/OS HCD Planning*, GA22-7525. After the production IODF is built, HCD displays a message.

If the work IODF has an activity log file defined for the work IODF, it is copied.
8.56 Production IODF created

After the production IODF has been built, HCD informs you that the production IODF has been created, as shown in Figure 8-57.

Production IODF created

After the production IODF has been built, HCD informs you that the production IODF has been created, as shown in Figure 8-57.
8.57 Activating a configuration with HCD

Activating a configuration with HCD

On the primary task selection panel, shown in Figure 8-10 on page 509, select **Activate or process configuration data**. On the resulting panel, select **Activate or verify configuration dynamically**. HCD displays the Activate or Verify Configuration panel.

Select what you want to activate. Figure 8-58 shows task 1. **Activate new hardware and software configuration** was selected here. (The panels for selecting the other tasks are similar.)

A configuration change is rejected if it includes a hardware delete for an I/O component that is online to the logical partition from which you are making the change. This is true even if you have entered YES in the Allow Hardware Deletes option field.

Therefore, you should vary offline any affected I/O component in all logical partitions. For example, when changing a channel path from unshared to shared, you must allow hardware deletes, and you must configure the channel path offline and vary offline the associated I/O devices before you activate the configuration.
8.58 View an active IODF with HCD

How to display active IODF from the HCD panels

HCD allows you to view the name and status of the IODF that has been used for IPL or for the last dynamic activation. The operating system configuration and EDT identifier and, if applicable, the configuration token, which is currently active in the hardware system area (HSA), are shown. Use the View Active Configuration function for an overview of the actual status for dynamic activation, indicating whether hardware and software changes are allowed.

On the primary task selection panel, shown in Figure 8-10 on page 509, select **Activate or process configuration data**, and then **View active configuration**.

The View Active Configuration panel is shown in Figure 8-59.
8.59 Viewing an active IODF

- **View active IODF from a system console**
  - `D IOS,CONFIG(ALL)`

  ![Example output]

  **How to display active IODF from an z/OS console**

  You can display the active IODF and HSA usage from a z/OS console with the following command, shown in Figure 8-60:

  `D IOS,CONFIG(ALL)`
### 8.60 Displaying device status

#### Display hierarchy

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D u,,,ddd,1</td>
<td>Is the device offline</td>
</tr>
<tr>
<td>D M=DEV(dddd)</td>
<td>What is the state of the paths to the device?</td>
</tr>
<tr>
<td>DS P,ddd</td>
<td>Are the paths physically available? (DASD or tape)</td>
</tr>
<tr>
<td>D M=CHP(cc)</td>
<td>What is the state of the paths on this CHPID?</td>
</tr>
<tr>
<td>D M=CHP</td>
<td>What is the state of the CHPID?</td>
</tr>
</tbody>
</table>

Figure 8-61 Displaying device status with z/OS commands

#### How to display device status

The recommended sequence of display commands to determine the status of a device is:

- **D U,,,ddd,1**  This command shows whether the device is online. No I/O operations are performed to present the display for this command. Therefore, the status displayed by the DISPLAY UNIT command shows the last known state of the device, and may not represent the actual physical status.

- **D M=DEV(dddd)**  If the device is not online, it may be necessary to bring online a path to the device. This command displays the channels that are defined to access the device, and the state of the paths over those channels. The output of this command may also display PATHS NOT VALIDATED. This text means that the path status to the displayed device has not been tested.

- **DS P,ddd**  This command displays the actual physical state of the paths to a DASD or tape device. If the path is not operational, then it is necessary to determine the state of the ESCON link and channel supporting the path.

- **D M=CHP(cc)**  This command displays the state of the paths to devices defined as accessible by this channel.

- **D M=CHP**  This command displays the state (online or offline) of the CHPID. Note that the display is relative to the logical partition supporting the operating system where the command is entered. Therefore, a channel shown as offline in this display may only be logically, not physically, offline.
8.61 HCD reports

Two styles of reports are available with HCD

- **Text**
  - CSS report
  - Switch report
  - OS report
  - CTC connection report
  - I/O path report (requires ESCON manager)
  - Compare IODF

- **Graphical**
  - LCU report
  - CU report
  - CHPID report
  - Switch report
  - CF connection report

Producing configuration reports

With HCD, you can create and print the following reports about the configuration data in an IODF:

- Channel Subsystem (CSS) Report
- Switch Report
- Operating System (OS) Report
- CTC Connection Report
- IODF Compare Report

These reports give you a printed overview of your configurations. You can create or build reports either with HCD panels or batch jobs.

Channel Subsystem Report

The Channel Subsystem Report contains all configuration data that is used by the channel subsystem. This consists of data, in summary and detail, about your servers, partitions, IOCDS, CHPIDs, switches, control units, and I/O devices.

If you have more than one server defined in your IODF, you can limit the report to the data for one server or partition. When limiting the report to one partition, only those channels are reported that have the designated partition in their access list. Likewise, only control units and devices that can be reached through these channels are reported.
Switch report
The Switch report contains details about your switch definition, the switch configurations for each switch, and port definitions.

If your IODF contains data for more than one switch, you can limit the report to the data for one switch and the configurations for that switch.

Operating System report
The Operating System report contains the configuration data that is used by the z/OS operating system. If your IODF contains data for more than one operating system, you can limit the report to the data for one operating system.

The Operating System report function can produce three principal types of reports: the Device Report, the EDT Report, and the Console Report.

- The Device Detail report contains data about devices and has two parts:
  - It contains detailed data about the devices.
  - It also contains summary data. The operating system summary report is not printed if you limit the OS device report to the data for one operating system.
- The EDT report contains data about all the EDTs of your configuration.
- The Console report contains data about all NIP consoles for z/OS.

CTC Connection report
The CTC Connection report shows CTC connections in your configuration that are defined through an ESCON Director. In the case of incorrect definitions, the report contains a list of messages with diagnostic information.

If the IODF contains more than one server or logical partition, you can limit the report to data for one server or logical partition.

Compare IODFs
You can use the Compare IODFs function to compare two IODFs and report the differences between them. For greater clarity, you can limit the compare reports to certain perspectives of the IODF:

- The Processor Compare report shows differences in the properties of partitions, CHPIDs, control units, and devices.
- The Switch Compare report shows differences in the properties of switches and switch configurations.
- The OS Configuration Compare report shows differences in device parameters, in features, in EDTs, in esoterics, in generics defined for EDTs, and consoles.

HCD graphical reports
It is possible with HCD to view and print graphical representations of your configuration. These may be stored in a data set for later printing, or viewed on a graphics-capable terminal.

Note: Prerequisite software is required for these functions. Refer to z/OS Hardware Configuration Definition: User's Guide, SC33-7988, for details.
Five graphical reports may be obtained as follows:

**LCU Report**  
Shows the CHPIDs, control units, and devices building one or more LCUs for a designated server.

**CU Report**  
Takes a control unit as focal point and shows all attachments of the control unit via switches up to the server. It also shows the devices attached to the control unit.

**CHPID Report**  
Shows, for a given server, all defined channel paths and what is attached to the CHPID (switches, CUs, and devices).

**Switch Report**  
Takes a switch (ESCON Director) as focal point and shows everything attached to the switch. Free ports of the switch are also shown. If the switch is connected to another switch, that switch is shown as well.

**CF Connect Report**  
Takes a Coupling Facility as focal point and shows all connections (CFS/CFR channel pairs) that exist between the Coupling Facility and the other servers defined in the IODF.

If “Include partitions” has been specified, the partitions are shown above each accessible CHPID.
8.62 Hardware Configuration Manager (HCM)

Hardware Configuration Manager (HCM) is an optional feature of z/OS, and extends the scope of configuration management provided by HCD.

HCM is a PC-based graphical user interface that allows you to easily navigate through the configuration diagrams and make changes in the configuration. HCM uses a client/server interface to HCD that combines the logical and physical aspects of z/OS hardware configuration management.

All updates to your configuration are done via the HCMs intuitive graphical user interface and most important, due to the client/server relationship with HCD, all changes of the logical I/O configuration are written into the IODF and fully validated and checked for accuracy and completeness by HCD, thus avoiding unplanned system outages due to incorrect definitions.

The logical information in the IODF represents the operating system and the channel subsystem definitions (the physical information cabinets, patch ports, crossbar switches, cables, locations, and so on), and this adds the infrastructure to the logical data. Furthermore, the logical and physical information for each object in the configuration match because they are created by the same process.

When you create an object, you add its logical and physical information at the same time. For example, when you connect a control unit to a server, the selected control units are logically defined to the selected CHPID through a control unit channel interface; the physical connection, including the cable, is displayed visually in the configuration diagram.
DS8000 series concepts

This chapter provides an overview of the concepts and rationale underlying DASD controllers, and then it specifically describes the IBM TotalStorage DS8000 Series.

The information in this chapter complements the “Storage management” section of ABCs of z/OS System Programming Volume 3, SG24-6983. That IBM Redbooks publication covers topics including RAID, Seascape architecture, Copy services, storage area network (SAN), and the major aspects of the IBM ESS. It also briefly focuses on the DS8000.

In this chapter, the following DS8000 topics are discussed:

- Characteristics
- Design
- Copy services
- Storage Hardware Management Console (S-HMC)
9.1 DASD controller capabilities

<table>
<thead>
<tr>
<th>Caching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple types of RAID</td>
</tr>
<tr>
<td>Multiple connections</td>
</tr>
<tr>
<td>Emulation of several control units</td>
</tr>
<tr>
<td>Logical partitioning</td>
</tr>
<tr>
<td>Copy services such as Point-in-time Copy and Remote Mirror and Copy</td>
</tr>
<tr>
<td>Multiple types of disks (capacity and rotation speed)</td>
</tr>
<tr>
<td>Multiple I/O requests ordered by z/OS priorities</td>
</tr>
<tr>
<td>Virtualization of DASD devices</td>
</tr>
<tr>
<td>Multiples I/Os in the same logical device</td>
</tr>
<tr>
<td>Possibility of hundreds of logical paths</td>
</tr>
</tbody>
</table>

Figure 9-1  DASD controller capabilities

DASD controller capabilities

There were no DASD controllers in the first mainframes. All the logic for storing data on DASD devices was kept in the devices. However, as DASD architecture emerged, the DASD controllers materialized. Today, a DASD controller globally has all the intelligence that previously existed in the devices, and much more.

The control unit accepts control signals from the channel, which controls the timing of data transfer over the channel path and provides indications concerning the status of the device. A typical operation is reading a recording medium and recording data. To accomplish its operations, the device needs detailed signal sequences peculiar to its type of device. The control unit decodes the commands received from the channel subsystem, interprets them for the particular type of device, and provides the signal sequence required for the performance of the operation.

A control unit may be housed separately, or it may be physically and logically integrated with the I/O devices or the channel subsystem.

A DASD controller in a sense is a true complex system, with the following aspects:

- Millions of software lines of code (known as Licensed Internal Code, or LIC)
- Several state-of-art RISC processors
- Huge real memory for data caching and internal LIC functions
- Complex internal fabric connections
- DASD space capacity in tens of terabyte units
Each DASD controller provides the following capabilities:

- Caching services to improve performance by minimizing disk accesses
- Arrays of inexpensive disks (RAID)
  Disk redundancy helps to avoid single-points-of-failure in the media that might cause data loss. There are several types of RAID implementations and usually the same DASD controller allows more than one type.
- Multiple connections, through host adapters, with distinct channel protocols such as FICON, ESCON, SCSI, FCP (by direct connection or through SAN by a switch). This implies storage sharing among multiple platforms such as Windows, UNIX instances, z/OS, and z/VM. A host adapter is a processor that is able to communicate with a channel to execute an I/O operation.
- Emulation of several control units, named logical control units, each with 256 logical devices
- Logical partitioning that allows splitting the DASD controller resources such as memory, processors, host adapters, and disk drives in logically independent and isolated controllers.
- Copy services such as Point-in-time Copy (for nondisruptive backups) and Remote Mirror and Copy (for business continuation)
- Multiple concurrent types of disks with different capacities and rotation speeds.
- Multiple I/O requests being processed, ordered in internal queues, and having z/OS-managed priorities.
- Virtualization of DASD devices (such as 3390s), where all software layers (including the operating system code) and channels have a logical view of the devices.
- Multiple I/O devices in the same logical device (PAV and multiple allegiance).
- Possibility of hundreds of connecting I/O channel instances through the implementation of logical paths.

A logical path is an internal DASD controller control block that represents an I/O channel instance that may communicate with such a controller. The same physical channel can have several instances to the controller, when this channel is being exploited by the multiple image facility; that is, the same physical channel serves several logical partitions (LPARs) on the server.
9.2 DS8000 characteristics

2-Way (Model 8100)
- Two dual-processor servers
- Up to 128GB Cache
- 8 to 64 200 MB/sec FC/FICON – 4 to 32 ESCON Ports
- 16 to 384 HDD - Intermixable 73GB
- 15Krpm, 146/300GB 10Krpm
- Physical capacity from 1.1TB up to 115TB

4-Way (Model 8300)
- Two four-processor servers
- Up to 256GB Cache
- 8 to 128 200 MB/sec FC/FICON – 4 to 64 ESCON Ports
- 16 to 640 HDD Intermixable 73GB
- 15Krpm, 146/300GB 10Krpm
- Physical capacity from 1.1TB up to 192TB

DS8000 models
The TotalStorage DS family (of which DS8000 is a component) is designed to offer high availability, multiplatform support and simplified management tools. It offers performance that is up to 6 times higher than the previous IBM ESS model 800. The storage capacity scales linearly from 1.1 TB up to 192 TB.

As shown in Figure 9-2, there are two models of the DS8000 DASD controllers, Model 8100 (submodel 921) and Model 8300 (submodels 922 and 9A2). Their characteristics are also listed in the figure.

DS8000 processors
The DS8000 has several types of processors. The most important ones (where the intelligence of the controller runs) have the latest pSeries® POWER5 processor technology, which has a 64-bit width. Model 8100 utilizes two 2-way symmetric multiprocessors with 1.5 GHz. The Model 8300 utilizes two 4-ways with 1.9 GHz. Each two-way and each four-way is called a complex.

There are also host adapters in charge of communicating with I/O channels in the upper level, and RAID device adapters interfacing with the disks. These processors are implemented with PCI-X processors running at a 64-bit width at 133 Mhz.
Cache
The DS8000 series offers up to 256 GB of volatile cache, which is up to 4 times as much as in previous ESS models. There is also a non-volatile cache (NVS) or persistent memory used to keep updated data generated by writes using “DASD fast write”. The cache implementation minimizes access to the disks. It acts like a store-in memory for the disks. Any cache miss for read forces a data movement from the disks. Any write only causes a movement to disks asynchronously when the cache is near to being full.

The contents of the NVS cache are kept nonvolatile due to the existence of internal batteries. However, because the contents of NVS are written to the internal SCSI disks of the DS8000 processor complex in case of a power failure, the contents of NVS can be preserved indefinitely. This means that unlike the DS6000 or ESS800, you are not held to a fixed limit of time before power must be restored.

Host adapters
Host adapters are processors in charge of communication with the I/O channels. The DS8000 offers four-port Fibre Channel/FICON host adapters. The 200 MBps Fibre Channel/FICON Express2 host adapters, which are offered in long-wave and shortwave (which affects the distance from the channel), can also auto-negotiate to 100 MBps link speeds. This flexibility enables immediate exploitation of the benefits offered by the higher performance, 200 MBps SAN-based solutions, while also maintaining compatibility with existing 100 MBps infrastructures.

In addition, the four ports on the adapter can be configured with an intermix of Fibre Channel Protocol (FCP) and FICON. The DS8000 also offers two-port ESCON adapters. A DS8000 can support up to a maximum of 32 host adapters, which provide up to 128 Fibre Channel/FICON ports. Host adapters reside in I/O enclosures.

RAID device adapters and disk drive modules DDMs
RAID device adapters are processors located in I/O enclosures in charge of accessing the disks. A DS8000 can have up to 16 RAID device adapters arranged into eight pairs. Each RAID device adapter has four ports.

The DS8000 offers a selection of disk drive modules (DDMs) and is an industry standard called Fibre Channels. Also, the protocol for the communication between the RAID device adapters and the DDMs is the switched Fibre Channel architected loop (FC-AL). There are up to three frames able to contain the DDMs. There are different types of DDMs:

- 73 GB with 15K revolutions per minute (RPM)
- 146 GB with 10K RPM or 15K RPM
- 300 GB with 10K RPM

They can be intermixed in the same DS8000 for better price performance.

Thus, there is a maximum limit of 640 DDMs, which in combination with the 300 GB drives gives a maximum capacity of 192 TB.
9.3 DS8000 design

Figure 9-3 DS8000 design

DS8000 design

Figure 9-3 shows the major elements of a DS8000 controller:

- The host adapters are the N-way SMP RISC processors. The memory for these processors is discussed in “DS8000 characteristics” on page 580.
- The higher bandwidth fault tolerant interconnect fabric is described in “Internal fabric and I/O enclosures” on page 583.
- The N-way SMP processors and the cache are described in “DS8000 characteristics” on page 580.
- LPAR concepts are described in “Logical partition (LPAR)” on page 593.
- The switched fabric to access the DDMs is presented in “Switched Fibre Channel Arbitrated Loop (FC-AL)” on page 586.
- Volatile memory and persistent memory are other names for the volatile cache and nonvolatile cache, respectively.
9.4 Internal fabric and I/O enclosures

**Internal fabric**
The internal fabric, also called higher bandwidth fault tolerant interconnect, uses an RIO-G (Remote I/O) protocol. This fabric is used to connect the POWER5 N-way SMP processors with host adapters and RAID device adapters. RAID device adapters and host adapters are installed in I/O enclosures, which also provides connectivity to them. There are eight I/O enclosures, each one having six slots. Two are used for RAID device adapters, the remaining slots are available to install up to four host adapters per I/O enclosure.

**Note:** Do not confuse I/O enclosures that contain adapters with disk enclosures that contain DDMs; see 9.5, “Disk subsystem” on page 584, for more information.

**RIO-G links**
RIO-G links are designed as a high performance self-healing interconnect. They can operate at 1 GHz frequency and offer a 2 GB per second sustained bandwidth per link.

There are two loops, the first involving four I/O enclosures and the second involving the other four I/O enclosures. In each loop, each POWER5 N-way SMP processor provides two external RIO-G ports, and each I/O enclosure provides another two.
9.5 Disk subsystem

Disk subsystem
Figure 9-5 shows the I/O enclosures containing the host adapters and the RAID device adapters connected by the RIO-G links. Also shown are the connections from the RAID device adapters to the DDMs.

The disk subsystem is made up of the following components:

- The RAID device adapters located in the I/O enclosures
  Each DS8000 device adapter (DA) card offers four 200 MBps FC-AL ports. These ports are used to connect the processor complexes to the disk enclosures. The adapter is responsible for managing, monitoring, and rebuilding the RAID arrays. The adapter provides remarkable performance thanks to a new high-function high-performance adapter called ASIC. A DS8000 can potentially have up to 16 of these adapters arranged into eight pairs.

- Disk enclosures containing the following:
  - Switched controller cards (also called SAN switches) that connect to the RAID device adapters
    This creates a switched Fibre Channel disk network. Each enclosure contains 20 such controller cards. Of these 20 ports, 16 are used to attach to the 16 disks in the enclosure and the remaining four are used to either interconnect with other enclosures or to the RAID device adapters. Each disk plugs into the disk enclosure backplane. The backplane is the electronic and physical backbone of the disk enclosure.
– Disks, commonly referred to as disk drive modules (DDMs)
   Each DS8000 disk enclosure contains a total of 16 DDMs.

**DS8000 frames**
Each DS8000 frame contains either eight disk enclosures (first frame) or 16 disk enclosures (expansion frames). Then, in the first frame, there is space for a maximum of 128 DDMs. Every expansion frame can contain 256 DDMs.

**DS8000 disk drives**
The physical capacity for the DS8000 is purchased though disk drive sets. A disk drive set contains 16 identical disk drives, which have the same capacity and the same revolution per minute (RPM).

Each DDM is reached by an industry standard FC-AL; see 9.6, “Switched Fibre Channel Arbitrated Loop (FC-AL)” on page 586.
9.6 Switched Fibre Channel Arbitrated Loop (FC-AL)

In the DS8000, the topology and communication protocol between pairs of RAID device adapters and DDMs is a switched FC-AL implementation. Switched FC-AL uses the standard FC-AL protocol, but the physical implementation is different. Before we explain switched FC-AL, we discuss the FC-AL topology first.

**FC-AL topology**

In a standard FC-AL disk enclosure, all disks are arranged in a loop. This loop-based architecture means that data flows through all disks before arriving at either end of the RAID device adapter.

Standard FC-AL access to DDMs presents the following issues:

- The full loop is required to participate in data transfer. Full discovery of the loop-by-loop initialization protocol (LIP) is required before any data transfer. Also, loop stability can be affected by DDM failures.
- In the event of a disk failure, it can be difficult to identify the cause of a loop breakage, leading to complex problem determination.
- There is a performance drop-off when the number of devices in the loop increases.
- To expand the loop it is normally necessary to partially open it. If mistakes are made, a complete loop outage can result.
Switched FC-AL offers a point-to-point connection to each drive and RAID device adapter. Figure 9-6 on page 586 shows that there are four paths available from each DDM to the pair of RAID device adapters. Each pair of RAID device adapters (with four ports each) accesses two device enclosures of 16 DDMs each. The key features of switched FC-AL technology are:

- Standard FC-AL communication protocol from DA to DDMs.
- Direct point-to-point links are established between DA and DDM.
- Isolation capabilities in case of DDM failures, providing easy problem determination.
- Predictive failure statistics.
- Simplified expansion; for example, no cable rerouting is required when adding another disk enclosure.

**Switched FC-AL advantages**

The DS8000 architecture employs dual redundant switched FC-AL access to each of the disk enclosures, which offers the following key benefits:

- Two independent networks to access the disk enclosures.
- Four access paths to each DDM.
- Each RAID device adapter port operates independently.

This is double the bandwidth of traditional FC-AL loop implementations with 250-500 MBps bandwidth to a RAID Rank. ESS was 40-250 MBps.

In Figure 9-6 on page 586, each DDM is depicted as being attached to two separate Fibre Channel switches. This means that with two device adapters, there are four effective data paths to each disk, with each path operating at 200 MBps.

When a connection is made between the RAID device adapter and a disk, the connection is a switched connection that uses an arbitrated loop protocol. This means that a mini-loop is created between the device adapter and the disk.

Whenever the RAID device adapter connects to a disk, it uses a switched connection to transfer data. This means that all data travels by the shortest possible path.

**DDM path redundancy**

Each DDM in the DS8000 is attached to two 20-switched controller card ports (SAN switches). These switches are built into the disk enclosure controller cards. Each DDM has two separate connections to the backplane, which allows it to be simultaneously attached to both switches.

If either disk enclosure controller card is removed from the enclosure, the switch that is included in that card is also removed. However, the switch in the remaining controller card retains the ability to communicate with all the disks and both device adapters (DAs) in a pair. Equally, each RAID DA has a path to each switch, so it also can tolerate the loss of a single path. If both paths from one DA fail, then it cannot access the switches. However, the other DA retains connection.
9.7 Redundant array of independent disks (RAID)

RAID architecture
Redundant array of independent disks (RAID) is a direct access storage implementation where, through redundancy, in the event of the loss of some disks there is no loss of data.

The original idea was to replace the physically big and sometimes reliable disks, like the 3390s, by many small computer system interface (SCSI) disks. The envisioned benefits of such an implementation are:

► Performance (due to parallelism).
► Cost (SCSIs are commodities).
► By implementing the logical disk concept (a kind of virtualization), different operating systems may share the same DASD controller.
► Environment (space and energy).

However, this approach increased the chances of malfunction due to media and disk failures and the fact that the logical device is now residing on many physical disks. The solution was to add redundancy to the design. By improving availability but wasting space (due to redundancy), this also caused performance problems such as “write penalty” and “free space reclamation.”

To address this performance issue, large caches are implemented in modern controllers, which decreases disk accesses. All the modern controllers now implement the RAID design. There are different types of RAID devices and different numbers of disks per array.
RAID description
The types of RAID, as certified by the RAID Architecture Board, are described briefly here:

**RAID-1**
Simply disk mirroring, like dual copy. Does not affect performance because it is done in parallel.

**RAID-5**
Refer to Figure 9-7 on page 588. There is one array of n disks (four in the figure). The three data physical blocks (record X, Y, and B) are logically connected to the same parity physical block stored in Volume 4. So, when X is written, the parity bits associated with X, Y, and B need to be updated as well.

In RAID-5 there is no specific volume in the array to contain the parity bits. In the example, for records W, V, and T, the parity bits are in Volume 2. The access arms move independently. Strong caching is necessary to avoid the write penalty; that is, four disk I/Os per write, namely two reads (old record contents plus old parity) and two writes (new record contents plus new parity). RAID-5 has a high read I/O rate for sequential processing because of natural striping.

RAID-5 performs the following tasks:

– It reads data from an undamaged disk. This is just one single disk I/O operation.
– It reads data from a damaged disk, which implies (n-1) disk I/Os, to recreate the lost data, where n is the number of disks in the array.
– For every write to an undamaged disk, RAID-5 performs four disk operations to store a correct parity block; this is called a write penalty. This penalty can be relieved with strong caching and a slice triggered algorithm (coalescing disks updates from cache into a single parallel I/O).
– For every write to a damaged disk, RAID-5 does n-1 reads and one parity write.

**RAID-3**
This looks like RAID-5, but in the array all the parity bits are written in just one disk. It is not used by modern controllers.

**RAID-6**
This has an array with two parity physical blocks and I/O requests in parallel with extra-record striping. Its access arms move independently (Reed/Salomon P-Q parity). The parity physical blocks are spread by the array disk. The write penalty is greater than RAID-5, with six disk accesses per write. However, you can survive without data loss with two disk failures in the same array.

**RAID-6+**
A RAID-6 without write penalty (due to log-structured file, or LFS), and has background free-space reclamation. The access arms all move together for writes. It is used by the RVA DASD controller.

**RAID-0**
There is no redundancy, only striping. In a sense this is not a type of RAID. The same 3390 logical volume is striped in several disks. Good performance for sequential reads and sequential writes because of parallelism.

**RAID-10**
This is a composition of RAID-0 (with striping) plus RAID-1 (with mirroring).
9.8 DS8000 types of RAID

DS8000 may have disk arrays RAID-5 (less expensive and less performance) or RAID-10 (more expensive and more performance) or both

RAID-5 arrays
- 6D+P
- 6D+P+S
- 7D+P

RAID-10 arrays
- 3D+S+3D+S
- 4D + 4D

Figure 9-8  DS8000 types of RAID

DS8000 types of RAID
The DS8000 can be configured as RAID-5, RAID-10, or a combination of both. For the two types of RAID, there is a need to allocate DDMs that constitute the RAID array or rank.

The DS8000 arrays are across loops (AAL). With AAL, an array site is actually split into two halves. Half of the site is located on the first disk loop of a DA pair and the other half is located on the second disk loop of that DA pair. It is implemented primarily to maximize performance.

However, in RAID-10 we can take advantage of AAL to provide a higher level of redundancy. The DS8000 Reliability Availability Serviceability (RAS) code deliberately ensures that one RAID-0 array is maintained on each of the two loops created by a DA pair. This means that in the extremely unlikely event of a complete loop outage, the DS8000 does not lose access to the RAID-10 array. This is because while one RAID-0 array is offline, the other remains available to service disk I/O.

RAID-5
RAID-5 offers an excellent price/performance ratio for many client applications (mainly for read only applications). RAID-10 can offer better performance at a higher price.
- A RAID-5 array contains seven or eight disks, depending on whether the array is supplying a spare, as explained here:
  - A 7-disk array uses one disk for parity, so it is referred to as a 6D+P array (where the P stands for parity).
– An 8-disk array with one spare, referred to as a 6D+P+S.
– An 8-8 disk array without spare, referred to as a 7D+P array.

When a disk drive module (DDM) fails in a RAID-5 array, the RAID device adapter starts an operation to reconstruct the data that was on the failed drive onto one of the spare drives. The spare that is used will be chosen based on a smart algorithm that looks at the location of the spares and the size and location of the failed DDM. The rebuild is performed by reading the corresponding data and parity in each stripe from the remaining drives in the array, performing an exclusive-OR operation to recreate the data, then writing this data to the spare drive.

While this data reconstruction is going on, the device adapter can still service read and write requests to the array from the hosts. There might be some degradation in performance while the sparing operation is in progress, because various DA and switched network resources are being used to do the reconstruction. Due to the switch-based architecture, this effect will be minimal. Additionally, any read request for data on the failed drive requires data to be read from the other drives in the array and then the DA performs an operation to reconstruct the data. The replacement of the failed disk will be the next spare.

RAID-10

In the DS8000, RAID-10 implementation is achieved using eight DDMs.
- If spares exist on the array site, then six DDMs are used to make a 3-disk RAID-0 array, which is then mirrored (3D + S + 3D + S).
- If spares do not exist on the array site, then eight DDMs are used to make a 4-disk RAID-0 array that is then mirrored (4D + 4D).

When a disk drive module (DDM) fails in a RAID-10 array, the RAID device adapter starts the reconstruction from the failed drive onto one of the hot spare drives. The spare that is used will be chosen based on a smart algorithm that looks at the location of the spares and the size and location of the failed DDM.

While this data reconstruction is going on, the DA can still service read and write requests to the array from the hosts. There might be some degradation in performance while the sparing operation is in progress, because some DA and switched network resources are being used to do the reconstruction. Due to the switch-based architecture of the DS8000, this effect will be minimal. Read requests for data on the failed drive will probably not be affected because they can all be directed to the good RAID-1 array. Performance of the RAID-10 array returns to normal when the data reconstruction onto the spare device completes.
9.9 Logical subsystem (LSS)

Logical subsystem

A logical subsystem (LSS) is a set of up to 256 logical volumes. In z/OS, LSS is also named logical control unit. This logical concept was introduced because in ESCON/ FICON protocols, each channel can refer to only 256 logical devices (volumes) per controller. Then, each DS8000 emulates up to 256 LSSs, to offer many more logical devices than just 256.

On the DS8000, it is up to the installation to associate a logical device address to an LSS. There is no fixed binding between any RAID rank (array) and any LSS. The capacity of one or more ranks can be aggregated into an extent pool. Extent pools belong to one DS8000 server (complex); that is, server 0 or server 1.

Logical volumes are configured into extent pools. Different logical volumes on the same logical subsystem can be configured in different extent pools but in the same server. Then, LSSs have an affinity to the servers.

- All even-numbered LSSs (X'00', X'02', X'04', up to X'FE') belong to server 0.
- All odd-numbered LSSs (X'01', X'03', X'05', up to X'FD') belong to server 1.

When creating logical volumes and assigning their logical volume numbers (also called device numbers), consider whether parallel access volumes (PAVs) are required on the LCU, and reserve several of the addresses on the LCU for alias addresses.
9.10 Logical partition (LPAR)

The DS8300 model 9A2 allows you to logically partition the system into two virtual storage system images with the following properties:

- Equal partitions for processors, cache, links, adapters, and disks between images.
- Robust isolation between images through hardware and PowerPC® hypervisor microcode (firmware); that is, one logical partition cannot take down another.
- Each logical partition can run its own level of code.

**Logical partition (LPAR)**

[Figure 9-10 Logical partition]

DS8300 Model 9A2 exploits LPAR technology, allowing to run two separate storage server images.

**Figure 9-10 Logical partition**

LUN 0
LUN 1
LUN 2

DS8000

LUN 0
Logical Partition A

LUN 0
Logical Partition B

LUN 2
Logical Partition A

LUN 2
Logical Partition B

Workload A

Workload B
9.11 Copy services classification criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Locality</th>
<th>Granularity</th>
<th>Timing for ending the copy</th>
<th>Purpose</th>
<th>Synchronously or asynchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Local</td>
<td>Data set level</td>
<td>Point-in-time (or T0 copy)</td>
<td>Fast backups</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remote</td>
<td>Logical volume level</td>
<td>Remote copy</td>
<td>Disaster recovery</td>
<td></td>
</tr>
</tbody>
</table>

Copy services classification

Copy services is a collection of functions that provide disaster recovery, data migration, and data duplication functions. It implies that the controller itself is able to copy data from one logical volume (3390, for example) to another logical volume.

With the Copy services functions, for example, you can create backup data with little or no disruption to your database executing transactions, and you can mirror your application data to a remote site for a disaster recovery implementation.

Many design characteristics of the DS8000 copy services features (data copying and mirroring capabilities) contribute to the protection of your data, 24 hours a day and seven days a week, implying a continuous availability (CA) environment.

Copy services criteria

Conceptually, we can classify the Copy services by different criteria, such as:

- **Locality**
  - Local, meaning that the primary and secondary logical volumes are located in the same DASD controller.
  - Remote, meaning that the primary and secondary volumes are not located in the same DASD controller, which can be any distance apart.
Granularity

- The copy is done at the data set level.
- The copy is done at the logical volume level.

Timing for ending the copy

- Point-in-time (or T0) copy, where the copy is made as a snapshot (freezing the time) and after that no further updated data in the primary volume is copied into the secondary volume.
- Remote copy, where the copy (mirroring) of the primary logical volume into the secondary theoretically never ends.

Purpose

- Fast backups
- Disaster recovery

Synchronous or asynchronous

- Synchronous, meaning that the application task that requires a write I/O on the primary logical volume waits until the copy into the secondary is completed.
- Asynchronous, meaning that the application task that requires a write I/O on the primary logical volume does not wait till the copy into the secondary is completed.

Agent

- The DASD controller is in charge of implementing the copy service.
- Software is in charge (together with the awareness of the controller) of implementing the copy services.
9.12 Consistency group concept

Sequence of I/Os in the source:
- 1. Write to log volume: Data Record #2 is being updated
- 2. Update Data Record #2 on data volume
- 3. Write to log volume: Data Record #2 update complete

Combinations of I/Os in the copy where data is consistent:
- Operation 1, 2, and 3
- Operation 1 and 2
- Operation 1

Combinations of I/Os in the copy where data is inconsistent:
- Operation 2 and 3
- Operation 1 and 3
- Operation 2
- Operation 3

Figure 9-12 Example of consistency group

Consistency group
Consistency group is a function to keep data consistency in the backup or mirrored copy. Data consistency means that the order of dependent writes is kept in the copy. In the following example, we have a database operation involving a log volume and a data volume:

1. Write to log volume: Data Record #2 is being updated.
2. Update Data Record #2 on data volume.
3. Write to log volume: Data Record #2 update complete.

If the copy of the data contains any of the following combinations, then the data is consistent, even if a data loss situation occurs:
- Operation 1, 2, and 3
- Operation 1 and 2
- Operation 1

If the copy of the data contains any of the following combinations, then the data is inconsistent (the order of dependent writes was not preserved):
- Operation 2 and 3
- Operation 1 and 3
- Operation 2
- Operation 3
Implementing consistency groups
To implement consistency groups, the controller must understand the sequence of the writes. If one of them could not be copied by a reason such as "extended long busy", then the next ones will be held. This extended long busy state is usually set by the controller when it cannot copy the data to the other controller.

A situation where the sequence of dependent writes cannot be copied in the same sequence is in a “rolling over disaster”, where due to fire or flood, the primary I/O controllers do not stop at the same time. The best disaster is always an explosion, as far as the sequence of dependent writes is concerned. Refer to 9.17, “Consistency groups in Metro Mirror” on page 605 and to 9.15, “Consistency group in FlashCopy” on page 602 for more information about the implementation of consistency groups in a DS8000.
9.13 Copy services in DS8000

- FlashCopy
- Remote Mirror and Copy (PPRC)
  - Metro Mirror (synchronous PPRC)
  - Global copy (PPRC extended distance)
  - Metro/Global mirror (asynchronous PPRC)
- Global Mirror for zSeries/z9 (XRC)
- Metro/Global Mirror for zSeries/z9

Copy services in DS8000

Figure 9-13 shows the types of Copy services available in DS8000. All of them are priced products. You can manage the Copy services functions through a command-line interface, the IBM TotalStorage DS CLI, and a new web-based interface, the IBM TotalStorage DS Storage Manager, as discussed in the following sections.
9.14 FlashCopy

FlashCopy® has the following Copy services properties: local, data set level or logical volume level, point-in-time, fast backup purpose, synchronous, and performed by the DS8000.

With FlashCopy, it instantaneously looks at T0 as the secondary copy is created. After T0, all the updates in the primary volume set are not propagated to the secondary because the target copy in the secondary volume must be coherent.

The point-in-time copy created by FlashCopy is typically used where you need a copy of the production data to be produced with little or no application downtime (depending on the application). The copy looks exactly like the original source volume and is almost instantly available in a binary copy mode. The copies produced by FlashCopy are used for:

- Data backups to be used in an eventual restore
- Data copies to be used in application development testing
- Data copies for data mining applications

FlashCopy operation setup

As shown in Figure 9-14, when you set up a FlashCopy operation, a relationship is established between the source and target pair of volumes and a bitmap of the source volume is created in seconds. In those seconds, no writes are to be allowed in the source.
After this relationship and bitmap are created, both volumes (primary and secondary) can be accessed (for reads and writes), as though all the data had been physically copied. Optionally, a background process physically copies the tracks from the source to the target logical volume.

If you access the source or the target volumes before the optional background copy is completed, then FlashCopy manages these I/O requests, as follows:

- **Read from the source volume**
  
  When you read some data from the source volume, it is simply read from the source volume.

- **Read from the target volume**
  
  When you read some data from the target volume, FlashCopy checks the bitmap. If the backup data is already copied to the target volume, it is read from the target volume. If the backup data is not copied yet, it is read from the source volume.

- **Write to the source volume**
  
  When you write data to the source volume, at first the updated data is written to the data cache and persistent memory (NVS cache) as usual. When the updated data is destaged to the source volume, FlashCopy checks the bitmap and performs the following action:
  
  - If the backup data is already copied, it is simply updated on the source volume.
  
  - If the backup data is not copied yet, first the backup data is copied synchronously to the target volume and after that it is updated on the source volume.

- **Write to the target volume**
  
  When you write data to the target volume, it is written to the data cache and NVS cache. FlashCopy manages the bitmaps to not overwrite this latest data (produced by this write) by the background process.

**No background copy option**

The physical background copy might have a slight impact on your application, because the physical copy needs various DS8000 storage resources. But the impact is minimal because the host I/O priority is larger than the background copy. However, if you want, you can issue FlashCopy with the “no background copy” option.

In this case the FlashCopy relationship is established without initiating a background copy. Therefore, you can minimize the impact of the background copy. The only case where we have a physical copy is when an update comes to a source track; then a copy of the point-in-time data is copied to the target volume so that it is available when the data from the target volume is accessed. This option is useful for clients that just want to take a backup copy of the target (usually to tape).

**FlashCopy options**

There are several FlashCopy options that an installation may use:

- **Refresh target volume (also known as Incremental FlashCopy)**
  
  This option provides the ability to refresh a target volume involved in a FlashCopy relationship. When a subsequent FlashCopy operation is initiated, only the tracks changed on both source and target need to be copied from source to the target. The direction of the refresh can also be reversed.

- **Data set FlashCopy**
  
  This option can be used instead of volume FlashCopy (only in z/OS).
Multiple relationship FlashCopy
This option allows a source to have FlashCopy relationships with multiple targets simultaneously.

Establish FlashCopy on existing Remote Mirror and Copy (PPRC) primary volume
This option allows you to establish a FlashCopy relationship where the target is also a remote mirror primary volume. This enables you to create full or incremental point-in-time copies at a local site and use remote mirroring commands to copy the data to the remote site.

Persistent FlashCopy
This option allows the FlashCopy relationship to remain even after the copy operation completes. You must explicitly delete the relationship.
9.15 Consistency group in FlashCopy

If a consistent point-in-time copy across many logical volumes is required, then you must use consistency group FlashCopy to create a consistent copy across multiple logical volumes in multiple storage controllers. See 9.12, “Consistency group concept” on page 596, for more information about this topic.

To create consistency groups in FlashCopy, it is necessary to have the possibility of freezing (temporarily queue) write I/O activity to a volume. The installation will issue a set of Establish FlashCopy commands with the freeze option, which will hold off host I/O to the source volumes. That is, consistency group FlashCopy provides the capability to temporarily queue (at the host I/O level, not the application level) subsequent write operations to the source volumes that are part of the consistency group. During the temporary queuing, Establish FlashCopy is completed and this condition is reset (allowing the write operations to resume) by the Consistency Group Created command or the time-out value expires (the default is two minutes).

In the event of an extended long busy situation, all the subsequent primary volumes in the consistency group are temporarily not copied.
9.16 Remote Mirror and Copy (example: PPRC)

Remote Mirror and Copy
The Remote Mirror and Copy feature can operate in the following modes: metro mirror (for example, sync PPRC), global copy (for example, PPRC-XD) and Global Mirror (for example, async PPRC). In all of these, the primary and the secondary volumes are not in the same controller. To create a framework to allow disaster recovery, it is desirable to put all the primary volumes in one controller (site).

The connection between the two controllers is through FCP links with an aggregate data rate of 200 MBps.

Metro Mirror (for example, sync PPRC)
Metro Mirror has the following copy services properties: remote, logical volume level, non point-in-time, disaster recovery predominantly, synchronous, and executed by the DS8000. See 9.11, “Copy services classification criteria” on page 594, for more information about this topic.

Metro Mirror provides real-time mirroring of logical volumes between two DS8000s that can be located up to 300 km from each other. It is a synchronous copy solution where write operations are completed on both copies (local and remote site) before they are considered to be complete and the waiting task is posted.

The primary controller receiving a write request from the channel moves the data to both caches and disconnects the channel (if ESCON), then sends the data to the secondary.
The secondary moves the data to both caches and sends an acknowledgement signal back to the primary. The primary sends a device end status to the channel. If this write is the last CCW in the channel program, it generates an I/O interrupt. Finally, IOS receives the I/O interrupt and posts the waiting task.

Theoretically, there is no data loss with Global Mirror, but there is a slight performance impact on the application task as indicated by a higher I/O disconnect time.

**HyperSwap**
The secondary volumes are not online to the z/OS systems reaching the secondary controller. Then, no writes and no reads are allowed. HyperSwap® provides the ability to nondisruptively swap from using the primary volume of a Metro Mirror pair to using what had been the secondary volume.

Prior to the availability of HyperSwap, an IPL was required on every system if you wanted to switch over to run off the secondary volumes. This meant that it was not possible to maintain application availability across a switch from primary to secondary volumes.

With HyperSwap, such a move can be accomplished with no IPL and with simply a brief pause in application availability. The HyperSwap function is designed to be completely controlled by automation, allowing all aspects of the site switch to be controlled through GDPS (a key product for handling disaster recovery problems). HyperSwap can be invoked in two ways:

- **Planned HyperSwap**
  This might be invoked in advance of planned disruptive maintenance to a control unit, for example. A planned HyperSwap is invoked manually using GDPS facilities.

- **Unplanned HyperSwap**
  An unplanned HyperSwap is invoked automatically by GDPS, triggered by events that indicate the failure of a primary disk device.

In both cases, the systems that are using the primary volumes will experience a temporary pause in processing. During this pause, the PPRC sessions are changed (mirroring may be established in the opposite direction, depending on the option selected). The UCBs for the primary devices are also updated to point to the former secondary volumes, and then the systems resume operation.
Chapter 9. DS8000 series concepts

9.17 Consistency groups in Metro Mirror

Global Mirror consistency group scenario

In this scenario we have two primary DASD controllers (1 and 3) in the main site connected through FCP links to two secondary DASD controllers (2 and 4, respectively) in the alternative site. The DB2 table spaces are in a logical volume in controller 1, and DB2 logs are in a volume in controller 2 (it is recommended that they be in different controllers). GDPS is installed in both sites.

When the number of modified pages in the DB2 buffer pools exceeds a certain threshold, DB2 orders a huge I/O operation to flush those buffer pools to the volume in controller 1. After that, a record is written in the DB2 log telling the end of the flushing. Those writes are dependent and DB2 forces the sequence by waiting between the first and the second write:

1. Write buffer
2. Wait
3. Write log

To avoid log information being invalid, the write in the log is not to come before the write in the database. By the way, DB2 is not aware that the Metro Mirror function is available.

When setting Metro Mirror copy service, the installation asks for CRITICAL=NO (as recommended); that is, if there is a failure in the FCP link or a failure in the secondary controller, the requesting task (DB2) is not informed of any problem, and the primary is updated.
As you can see, even with Metro Mirror there is a chance of data loss. The primary controller keeps the information about all tracks changed and not mirrored. As soon as the failure is corrected, these changes are copied in the background to the secondary controller, thus reestablishing the synchronism.

Disaster scenario
Now let us imagine a disaster scenario (without GDPS in the primary site and without a controller supporting consistence groups), where a rolling-over disaster initially only destroys the link between controller 1 and its secondary (2). Due to the CRITICAL=NO option, there is no mirroring of DB2 data and the DB2 task is posted normally. Getting out of the wait state, the DB2 task writes to the log the information about the ending of the flush and in this moment the FCP link between the primary controller 3 and its secondary (4) is still up; remember that it is a rolling-over disaster. The bottom line is that in the secondary site the log tells that the flush occurred and the database does not have the updates; that is, the integrity was totally lost.

Avoiding that disaster scenario
To avoid that scenario, we need the concept of consistency group and GDPS running in the primary site. Here is the same scenario, but with GDPS up and with DS8000.

When the mirroring in controller 1 is broken, the controller generates an I/O interrupt with an extended long busy status. This interrupt is treated by IOS, which issues a specific message to the z/OS console. This message is captured by GDPS automation. GDPS communicates with controller 1 to freeze the write I/O operation. “Freeze” here means that the controller will not send (holds) the “device end” I/O interrupt that will post the DB2 task.

Now, GDPS knows that the log volume (in 3) belongs to the same consistency group as the data base volume (in 1). GDPS communicates with controller 3, asking for a break in the mirroring of the log volume; that is, to cancel the Metro Mirror of the log volume in controller 3.

After that, GDPS takes the database volume in controller 1 from the frozen state. As a consequence, the “device end” I/O interrupt is generated, getting DB2 out of the wait state. Now DB2 issues the write to the log. However, this write will not be mirrored by Metro Copy and hence the integrity of the database is guaranteed in the second site. As you can see, there is data loss, but no loss of integrity.

For completeness, the possible functions executed by GDPS in the survival site include those listed here:

- CBU activation.
- Logical partition activation.
- IPL some z/OS systems.
- Execute HyperSwap.
- Activate a new I/O configuration.
- Ensure that secondary PPRC volumes and CF structures are time-consistent.
- Start new address spaces.
- Catalog data sets.
- Start monitors.
- Restore backed-up files.
Global Copy (example: PPRC XD)

9.18 Global Copy (example: PPRC XD)

Figure 9-18  Global Copy

Global Copy

Global Copy has the following copy services properties: remote, logical volume level, non-point-in-time, data migration predominantly, non-synchronous, and executed by the DS8000. See 9.11, “Copy services classification criteria” on page 594, for more information about this topic.

Global Copy is an asynchronous remote copy function for z/OS and open systems for longer distances than are possible with Metro Mirror. With Global Copy, write operations complete on the primary storage system before they are received by the secondary storage system, so the I/O disconnect time is not increased. This capability is designed to prevent the primary system’s performance from being affected by wait time from writes on the secondary system. Therefore, the primary and secondary copies can be separated by any distance.

This function is appropriate for remote data migration, offsite backups, and transmission of inactive database logs at virtually unlimited distances. When operating in Global Copy mode, the primary controller sends a periodic, incremental copy of updated tracks of the source volume to the target volume instead of a constant stream of updates. This causes less impact to application writes for source volumes and less demand for bandwidth resources, while allowing more flexible use of the available bandwidth.

Global Copy does not keep the sequence of write operations. Therefore, the copy is normally fuzzy. See 9.19, “Global Mirror (example: async PPRC)” on page 608, for information about how to avoid that situation.
9.19 Global Mirror (example: async PPRC)

Global Mirror has the following copy services properties: remote, logical volume level, time driven point-in-time, disaster recovery predominantly, synchronous/non-synchronous, and executed by the DS8000. See 9.11, “Copy services classification criteria” on page 594, for more information about this topic.

Global Mirror provides a long-distance remote copy feature across two sites using asynchronous technology. This solution is based on the existing Global Copy and FlashCopy. With Global Mirror (as in Global Copy), the data that the host writes to the storage unit at the local site is asynchronously shadowed to the storage unit at the remote site.

At a certain point in time, a consistency group is created using all of the primary volumes (shown as A in Figure 9-19) even if they are located in different DS8000 controllers. This has no application impact because the creation of the consistency group is fast (on the order of milliseconds).

After the consistency group is created, the application writes can continue updating the primary volumes. Global Mirror goes-to-sync and the increment of the consistent data is sent synchronously to the secondary volumes (B) using the existing Global Copy relationship. After the data reaches the B volumes, it is FlashCopied to the C volumes; see Figure 9-19.

The C volumes now contain the consistent copy of the data. Because the B volumes usually contain a fuzzy copy of the data from the local site (not when doing the FlashCopy), the C volumes are used to hold the last point-in-time consistent data while the B volumes are being
updated by the Global Copy relationship. After establishment of the FlashCopy, you can change the Remote Mirror and Copy (PPRC) mode back to the non-synchronous mode.

When you implement Global Mirror copy services, you set up the FlashCopy between the B and C volumes with options *no background copy* and *start change recording*. It means that before the latest data is updated to the B volumes, the last consistent data in the B volume is moved to the C volumes. Therefore, at some time, a part of consistent data is in the B volume, and the other part of consistent data is in the C volume. If a disaster occurs during the FlashCopy of the data, special procedures are needed to finalize the FlashCopy.

**Note:** Remote Mirror and Copy (PPRC) can do failover and failback operations.

- A failover operation is the process of temporarily switching production to a backup facility (normally your recovery site) following a planned outage (such as a scheduled maintenance period) or an unplanned outage (such as a disaster).
- A failback operation is the process of returning production to its original location.

These operations use Remote Mirror and Copy functions to help reduce the time required to synchronize volumes after the sites are switched during a planned or unplanned outage.

**Global Mirror operations**

Global Mirror operations provide the following benefits:

- Support for virtually unlimited distances between the local and remote sites, with the distance typically limited only by the capabilities of the network and the channel extension technology. This unlimited distance enables you to choose your remote site location based on business needs, and enables site separation to add protection from localized disasters.

- A consistent and restartable copy of the data at the remote site, created with minimal impact to application performance at the local site.

- Data currency (where, for many environments, the remote site lags behind the local site typically by 3 to 5 seconds), thus minimizing the amount of data exposure in the event of an unplanned outage. The actual lag in data currency that you experience can depend upon a number of factors, including specific workload characteristics and bandwidth between the local and remote sites.

- Dynamic selection of the desired recovery point objective, based upon business requirements and optimization of available bandwidth.

- Session support, whereby data consistency at the remote site is internally managed across up to eight storage units that are located across the local and remote sites.

- Efficient synchronization of the local and remote sites with support for failover and failback modes, helping to reduce the time that is required to switch back to the local site after a planned or unplanned outage.
9.20 z/OS Global Mirror (example: XRC)

z/OS Global Mirror has the following copy services properties: remote, logical volume level, no point-in-time, disaster recovery predominantly, non-synchronous, and executed by the z/OS DFSMS component system data mover (SDM). Refer to 9.11, “Copy services classification criteria” on page 594, for more information about this topic.

All the copy activity is done by SDM, that is, reading from primary volumes and writing into secondary volumes. There is no link between primary and secondary DS8000. The secondary controller does not need any special feature and can be any DASD controller. SDM may run in a local or a remote system, so it clearly needs channel connections (including channel extensions) to all primary and secondary controllers.

The z/OS Global Mirror function mirrors data on the storage unit to a remote location for disaster recovery. It protects data consistency across all volumes that you have defined for mirroring. The volumes (primary and secondary) can reside on several different DASD controllers. The z/OS Global Mirror function can mirror the volumes over several thousand kilometers from the source site to the target recovery site (as for any non-synchronous copy services).

With z/OS Global Mirror, you can suspend (not terminate) or resume (restart) copy services during an outage. Only data that changed during the outage needs to be resynchronized between the copies.

Figure 9-20  z/OS Global Mirror
A chronology of events covering z/OS Global Mirror, showing how the sequence of depending writes is guaranteed, follows:

T0: IOS in the primary server adds a time stamp to a physical record to be written. The SSCH is issued towards a logical 3390 volume in DS8000 A, and the requesting task is in wait state.

DS8000 A keeps a physical block with the time stamp in both caches and returns device end to the channel. The channel generates an I/O interrupt, and IOS posts the requesting task, indicating the end of the I/O operation.

T1: Several sequences of the T0 events happen in DS8000 B in primary logical 3390 volumes.

T2: Several sequences of the T0 events happen in DS8000 A in primary logical 3390 volumes.

T3: SDM gets the control running in the secondary server, and asks (through channel connection) for all updates (with the time stamps) from DS8000 A and B.

T4: SDM classifies all the updates per volume by time stamp. A cut-off time stamp is determined. It is the least recent of the two most recent time stamps in A and B updates. All time stamps older than the cutoff will not be copied to the secondary volumes; they will be in the next SDM cycle. This procedure guarantees the sequence of all writes in the secondary volumes.

If you have more than one system running in different servers in the primary site, a Sysplex Timer is mandatory for time stamp coherency.
9.21 Parallel access volume (PAV)

Parallel access volume (PAV)
The z/OS system Input Output Supervisor (IOS) maps a device in a unit control block (UCB). Traditionally this I/O device does not support concurrency, being treated as a single resource, serially used. Then, high I/O activity towards the same device can adversely affect performance. This contention is worse for large volumes with many small data sets. The symptom that is displayed is extended IOSQ time, where the I/O request is queued in the UCB. z/OS cannot attempt to start more than one I/O operation at a time to the device.

The ESS and DS8000 support concurrent data transfer operations to or from the same 3390/3380 devices from the same system. A device (volume) accessed in this way is called a parallel access volume (PAV).

PAV exploitation requires both software enablement and an optional feature on your controller. PAV support must be installed on each controller. It enables the issuing of multiple channel programs to a volume from a single system, and allows simultaneous access to the logical volume by multiple users or jobs. Reads, and writes to different extents, can be satisfied simultaneously. The domain of an I/O consists of the specified extents to which the I/O operation applies, that corresponds to the extents of the same data set. Writes to the same domain still have to be serialized to maintain data integrity, and so it is for reads and write.

The implementation of N parallel I/Os to the same 3390/3380 device consumes N addresses in the logical controller, then decreasing the number of possible real devices. Also, UCBs are
not prepared to allow multiples I/Os due to software products compatibility issues. Support is then implemented by defining multiple UCBs for the same device.

The UCBs are of two types:

- **Base address**
  This is the actual unit address. There is only one for any volume.

- **Alias address**
  These addresses are mapped back to a base device address. I/O scheduled for an alias is executed against the base by the controller. No physical disk space is associated with an alias address. Alias UCBs are stored above the 16 MB line.

**PAV benefits**

Workloads that are most likely to benefit from PAV function being available include:

- Volumes with many concurrently open data sets, such as volumes in a work pool
- Volumes that have a high read-to-write ratio per extent
- Volumes reporting high IOSQ times

Candidate data sets types are:

- High read-to-write ratio
- Many extents on one volume
- Concurrently shared by many readers
- Accessed using media manager or VSAM-extended format (32-byte suffix)

PAVs can be assigned to base UCBs either:

- Manually (static) by the installation
- Dynamically - WLM can move alias UCBs from one base UCB to another base UCB to:
  - Balance device utilizations.
  - Honor the goal of transactions suffering I/O delays because of long IOSQ time. All WLMs in a sysplex must agree with the movement of alias UCBs.

The following differences are addressed by the HyperPAV implementation:

- Any change must be decided by all WLMs in a sysplex using XCF communication.
- The number of aliases for one device must be equal in all z/OS systems.
- Any change implies a dynamic I/O configuration.

To solve such problems HyperPAV is introduced, where all alias UCBs are located in a pool and are used dynamically by IOS.


9.22 HyperPAV feature for DS8000 series

- **HyperPAV - New feature of PAV Volumes in DS8000**
  - Reduces the number of PAV-aliases needed per logical subsystem (LSS)
    - By an order of magnitude but still maintaining optimal response times
  - This is accomplished by no longer statically binding PAV-aliases to PAV-bases
    - WLM no longer adjusts the bindings
    - In HyperPAV mode, PAV-aliases are bound to PAV-bases only for the duration of a single I/O operation, thus reducing the number of aliases required per LSS significantly

*Figure 9-22 HyperPAV implementation*

**DS8000 feature**

HyperPAV is an optional feature on the DS8000 series, available with the HyperPAV indicator feature number 0782 and corresponding DS8000 series function authorization (2244-PAV HyperPAV feature number 7899). HyperPAV also requires the purchase of one or more PAV licensed features and the FICON/ESCON Attachment licensed feature. (The FICON/ESCON Attachment licensed feature applies only to the DS8000 Turbo Models 931, 932, and 9B2). HyperPAV allows many DS8000 series users to benefit from enhancements to PAV with support for HyperPAV. HyperPAV allows an alias address to be used to access any base on the same control unit image per I/O base. This capability also allows different HyperPAV hosts to use one alias to access different bases, which reduces the number of alias addresses required to support a set of bases in a System z environment with no latency in targeting an alias to a base. This functionality is also designed to enable applications to achieve equal or better performance than possible with the original PAV feature alone, while also using the same or fewer z/OS resources. The HyperPAV capability is offered on z/OS V1R6 and later.

**HyperPAV implementation**

HyperPAV allows an alias address to be used to access any base on the same control unit image per I/O base. Such capability also allows different HyperPAV hosts to use one alias to access different bases. This reduces the number of alias addresses required to support a set of bases in a System z environment with no latency in targeting an alias to a base. This functionality is also designed to enable applications to achieve equal or better performance
than possible with the original PAV feature alone, while using the same or fewer z/OS resources.
HyperPAV and IOS

The person depicted in Figure 9-23 represents the role of the z/OS component Input Output Supervisor (IOS). The graphic shows that there is an outstanding I/O operation towards device number 0280. Its only UCB (base) is busy due to an ongoing I/O operation. So, IOS goes to a “box” of free UCB aliases and picks one of them. This alias UCB binds to the base device and the I/O is started successfully. When the I/O operation finishes, the alias UCB is returned to the box by IOS.

As shown, all the devices shown with an alias UCB have the base and the alias UCB in a busy state. A device can be assigned as many alias UCBS as needed.
9.24 HyperPAV implementation

New HyperPAV scheme

With the IBM System Storage DS8000 Turbo model and the IBM server synergy feature, the HyperPAV together with PAV, Multiple Allegiance, and support for IBM System z MIDAW facility can dramatically improve performance and efficiency for System z environments.

In 9.24, “HyperPAV implementation” on page 617, you see a different number of UCB aliases in the same device in different z/OS systems. The UCB Alias pool containing free UCBs is also depicted. In HyperPAV mode, PAV-aliases are no longer statically bound to PAV-bases, but instead are bound only for the duration of a single I/O operation, thus reducing the number of aliases required for an LCU. This offers several advantages, such as an increase of available storage space without change of device geometry, and an elimination of multisystem alias management overhead.

With HyperPAV technology:

- z/OS uses a pool of UCB aliases.
- As each application I/O is requested, if the base volume is busy with another I/O, the following actions occur:
  - z/OS selects a free alias from the pool, quickly binds the alias device to the base device, and starts the I/O.
  - After the I/O completes, the alias device is used for another I/O on the LSS or is returned to the free alias pool.
If too many I/Os are started simultaneously, then the following actions will occur:
- z/OS will queue the I/Os at the LSS level.
- When an exposure frees up that can be used for queued I/Os, they are started.
- Queued I/O is done within an assigned I/O priority.

For each z/OS image within the sysplex, aliases are used independently:
- WLM is not involved in alias movement so it does not need to collect information to manage HyperPAV aliases.

**HyperPAV implementation**
HyperPAV is implemented in the parmlib member as follows:

```
SYS1.PARMLIB(IECIOStt) with HYPERPAV=YES|NO|BASEONLY
```

Where:
- **YES** means attempt to initialize LSSes in HyperPAV mode.
- **NO** means do not attempt to initialize LSSes in HyperPAV mode.
- **BASEONLY** means attempt to initialize LSSes in HyperPAV mode, but only start I/Os on base volumes.

**Enhanced z/OS commands**
The following commands are enhanced:
- **SETIOS HYPERPAV=**YES | NO | BASEONLY
- **SET IOS=**xx
- **D M=**DEV
- **D IOS, HYPERPAV**
- **DEVSERV QPAV, dddd**
9.25 Display M=DEV command

Figure 9-25  Display M=DEV command

d m=dev(0710)
IEE174I 23.35.49 DISPLAY M 835
DEVICE 0710   STATUS=ONLINE
CHP       10   20   30   40
DEST LINK ADDRESS 10   20   30   40
PATH ONLINE Y   Y   Y   Y
CHP PHYSICALLY ONLINE Y   Y   Y   Y
PATH OPERATIONAL Y   Y   Y   Y
MANAGED    N   N   N   N
CU NUMBER  0700 0700 0700 0700
MAXIMUM MANAGED CHPID(S) ALLOWED: 0
DESTINATION CU LOGICAL ADDRESS = 07
SCP CU ND   = 002107.000.IBM.TC.03069A000007.00FF
SCP TOKEN NED = 002107.900.IBM.TC.03069A000007.0700
SCP DEVICE NED = 002107.900.IBM.TC.03069A000007.0710
HYPERPAV ALIASES IN POOL  4

Figure 9-25  Display M=DEV command

Display M=DEV command

Figure 9-25 shows the output of this command. Note that there are four aliases in the pool.

You can detect this condition by outstanding message IOS166E or by using the D
M=DEV(bbbb) command, where bbbb is a base device in the logical control unit. MVS can use
the HyperPAV aliases only after the channels that are not FICON channels are removed from
the HyperPAV configuration, or these channels are configuring offline. However, it might be
necessary under this or other conditions to force MVS to rediscover aliases that were not
discovered previously.

Use the VARY bbbb,ONLINE,UNCOND command where bbbb is an online base device in the
logical control unit. You can use the VARY bbbb,ONLINE,UNCOND command to correct problems
when HyperPAV aliases are not properly used. Such an error condition occurs when
channels that are not FICON channels are configured online to a HyperPAV-capable control
unit. Because MVS operates on base devices within that logical subsystem in base only
mode, MVS cannot use alias devices for bases in the logical control unit.
9.26 RMF DASD report

As shown in 9.26, “RMF DASD report” on page 620, the PAV column now contains an H, which indicates that HyperPAV is active in the volume.
9.27 RMF I/O Queueing report

There are two new fields in this report connected with HyperPAV: HyperPAV Wait, and HyperPAV Max, as explained here.

**HyperPAV Wait**

This field indicates:

- The ratio of the number of times an I/O could not start and the total number of I/O requests.
- The ratio of HyperPAV Wait might indicate that the pool of alias devices was not large enough.
- An expanding pool can reduce I/O queue time and thus, CPU overhead.

**HyperPAV Max**

This field indicates:

- The maximum number of concurrently in-use HyperPAV alias devices for that LCU within that interval.

In this case, HyperPAV Max approaches the number of configured HyperPAV alias devices for the LSS pool for the peak intervals, so you can reduce the I/O queue time by adding more alias devices to the LSS pool.

---

**Figure 9-27   RMF I/O Queuing report**

<table>
<thead>
<tr>
<th>Storage</th>
<th>Device</th>
<th>Volume</th>
<th>LCU</th>
<th>Activity</th>
<th>Resp</th>
<th>IOSQ</th>
<th>CMR</th>
<th>DLY</th>
<th>CMR DLY Time</th>
<th>Time</th>
<th>DLY Time</th>
<th>CMR</th>
<th>DISC</th>
<th>Conn</th>
<th>Dev</th>
<th>Dev</th>
<th>Number</th>
<th>Any</th>
<th>MT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Num</td>
<td>Type</td>
<td>Serial</td>
<td>Rate</td>
<td>Time</td>
<td>DLY</td>
<td>Conn</td>
<td>UTIL</td>
<td>RESV</td>
<td>ALLOC</td>
<td>PEND</td>
<td>THRASH1 1000 33903</td>
<td>MM1000 1.0H 000D</td>
<td>0.092</td>
<td>0.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>System</td>
<td>ID</td>
<td>DATE</td>
<td>INTERVAL</td>
<td>Time</td>
<td>CYCLE</td>
<td>1.000 SECONDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL SAMPLES</td>
<td>899</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IODF</td>
<td>CR-DATE: 03/19/2006</td>
<td>CR-TIME: 22.47.41</td>
<td>ACT: ACTIVATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9.28 DS8000 Capacity on Demand

- DS8000 series can use the priced IBM Standby Capacity on Demand (CoD) option:
  - Standby CoD disk set contains 16 disk drives of the same capacity and RPM (10,000 or 15,000).
  - Activate by a nondisruptive activity that does not require intervention from IBM.
  - Upon activation of any portion of the Standby CoD disk drive set, you must place an order with IBM to initiate billing for the activated set.

Figure 9-28  DS8000 Capacity on Demand (CoD)

Capacity on Demand
To further help meet the changing storage needs of growing businesses, the DS8000 series can use the IBM Standby Capacity on Demand option (CoD), which is designed to allow clients to access extra capacity quickly whenever the need arises.

A Standby CoD disk set contains 16 disk drives of the same capacity and RPM (10000 or 15000). With this offering, up to four Standby CoD disk drive sets (64 disk drives) can be factory-installed or field-installed into your system.

To activate, you logically configure the disk drives for use. This is a nondisruptive activity that does not require intervention from IBM; simply download a software key. Upon activation of any portion of the Standby CoD disk drive set, you must place an order with IBM to initiate billing for the activated set. At that time you can also order replacement Standby CoD disk drive sets.
9.29 DS command line interface (CLI)

IBM TotalStorage DS command line interface (CLI)

The DS CLI is a single CLI that has the ability to perform a full set of commands for logical configuration and Copy Services activities. It is now possible to combine the DS CLI commands into a script. This can enhance your productivity because it eliminates the previous requirement to create and save a task using the GUI. The DS CLI can also issue Copy Services commands to an ESS Model 750, ESS Model 800, or DS6000 series system.

The following list highlights a few of the specific types of functions that you can perform with the DS command line interface:

- Check and verify your storage unit configuration
- Check the current Copy Services configuration that is used by the storage unit
- Create new logical storage and Copy Services configuration settings
- Modify or delete logical storage and Copy Services configuration settings

Storage management

ESS CLI commands that are used to perform storage management on the ESS 800, are issued to a process known as the *infoserver*. An *infoserver* runs on each cluster, and either *infoserver* can be used to perform ESS 800 storage management. Storage management on the ESS 800 will continue to use ESS CLI commands. Storage management on the DS8000 will use DS CLI commands.
9.30 Storage Hardware Management Console (S-HMC)

S-HMC
The DS8000 offers a new integrated management console. This console is the service and configuration portal for up to eight DS8000s in the future. Initially there will be one management console for one DS8000 storage subsystem. The S-HMC is the focal point for configuration and Copy Services management, which can be done by the integrated keyboard display or remotely through a web browser.

In the entirely new packaging there are also new management tools such as the DS Storage Manager and the DS command line interface (CLI), which allow for the management and configuration of the DS8000 series and the DS6000 series.

The DS CLI software must authenticate with the S-HMC or CS Server before commands can be issued. An initial setup task will be to define at least one user ID and password and save the authentication details in an encrypted file. A profile file can then be used to identify the name of the encrypted password file. Scripts that execute DS CLI commands can use the profile file to get the password needed to authenticate the commands.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 626. Note that some of the documents referenced here may be available in softcopy only.

- IBM eServer zSeries 990 Technical Introduction, SG24-6863
- IBM System z Connectivity Handbook, SG24-5444
- IBM System z9 Enterprise Class Technical Guide, SG24-7124
- IBM eServer zSeries 990 Technical Guide, SG24-6947
- Technical Introduction: IBM eServer zSeries 800, SG24-6515
- z/OS Intelligent Resource Director, SG24-5952
- ABCs of z/OS System Programming Volume 11, SG24-6327

Other publications

These publications are also relevant as further information sources:

- z/Architecture Reference Summary, SA22-7871
- z/Architecture Principles of Operations, SA22-7832
- z/OS Hardware Configuration Definition Planning, GA22-7525
- z/OS Distributed File Service zSeries File System Administration, SC24-5989
- OSA-Express Customer's Guide and Reference, SA22-7935

Online resources

These Web sites and URLs are also relevant as further information sources:

- Fibre Channel standards
  http://www.t10.org
  http://www.t11.org
- zSeries I/O connectivity
  http://www.ibm.com/servers/eserver/zseries/connectivity
- Parallel Sysplex
  http://www.ibm.com/servers/eserver/zseries/pso
- zSeries networking
  http://www.ibm.com/servers/eserver/zseries/networking
- IBM documentation and tools
  http://www.ibm.com/servers/resourcelink
- The IBTA is responsible for compliance testing of commercial products, a list of which can be found at:
  http://www.infinibandta.org/itinfo/IL

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

  ibm.com/redbooks
ABCs of z/OS System Programming
Volume 10

IBM System z processor design and connectivity
z/Architecture, LPAR concepts, HCD, z196, z10
DS8000 DASD controller

The ABCs of z/OS System Programming is an 13-volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you would like to become more familiar with z/OS in your current environment, or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection will serve as a powerful technical tool.

The contents of the volumes are as follows:
Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation
Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLST, authorized libraries, SMP/E, Language Environment
Volume 3: Introduction to DFSMS, data set basics storage management hardware and software, catalogs, and DFSMSds
Volume 4: Communication Server, TCP/IP, and VTAM
Volume 5: Base and Parallel Sysplex, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), Geographically Dispersed Parallel Sysplex (GDPS)
Volume 6: Introduction to security, RACF, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries firewall technologies, LDAP, and Enterprise identity mapping (EIM)
Volume 7: Printing in a z/OS environment, Infoprint Server and Infoprint Central
Volume 8: An introduction to z/OS problem diagnosis
Volume 9: z/OS UNIX System Services
Volume 10: Introduction to z/Architecture, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and HMC
Volume 11: Capacity planning, performance management, RMF, and SMF
Volume 12: WLM
Volume 13: JES3

SG24-6990-04
ISBN 073843650X

For more information: ibm.com/redbooks