

# ABCs of z/OS System Programming Volume 10

Alvaro Salla

Patrick Oughton







International Technical Support Organization

**ABCs of z/OS System Programming Volume 10**

May 2018

**Note:** Before using this information and the product it supports, read the information in “Notices” on page vii.

**Sixth Edition (May 2018)**

This edition applies to version 2 release 3 of IBM z/OS (product number 5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

**© Copyright International Business Machines Corporation 2017, 2018. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	vii
Trademarks .....	viii
<b>Preface</b> .....	ix
The team who wrote this book .....	ix
Now you can become a published author, too! .....	x
Comments welcome .....	x
Stay connected to IBM Redbooks .....	x
<b>Chapter 1. Introduction to z/Architecture</b> .....	1
1.1 z/Architecture historical path .....	2
1.2 Computer architecture overview .....	3
1.3 Concept of a process .....	5
1.4 System components .....	7
1.4.1 System Assist Processor (SAP) .....	8
1.4.2 Channels .....	9
1.4.3 Dynamic Switch .....	9
1.4.4 IBM Virtual Flash Memory (VFM) .....	9
1.4.5 Pervasive encryption .....	9
1.5 Processing units (PUs) .....	10
1.6 Microcode Concepts .....	11
1.7 z/Architecture Components .....	12
1.8 PU Registers .....	13
1.8.1 General purpose registers (GPR) .....	14
1.8.2 Floating point registers .....	15
1.9 Instruction set and its formats .....	16
1.10 Program status word (PSW) .....	17
1.10.1 PSW format .....	18
1.11 Multiprocessing .....	24
1.12 z/Architecture Data Formats .....	24
1.12.1 Binary integers .....	25
1.12.2 Decimal numbers .....	25
1.12.3 Floating-point numbers .....	26
1.12.4 EBCDIC data .....	26
1.12.5 ASCII data .....	27
1.12.6 Unicode .....	27
1.12.7 UTF-8 .....	27
1.13 IPL of a z/OS logical partition .....	27
1.14 CPC memory addressing .....	27
1.15 Addresses and address spaces .....	28
1.16 Addressing mode .....	30
1.16.1 AMODE .....	30
1.16.2 RMODE .....	31
1.17 Interrupts .....	31
1.17.1 Reasons for interrupts .....	33
1.17.2 Types of interrupts .....	33
1.17.3 Interrupt processing .....	37
1.18 Prefix Storage Area (PSA) .....	38
1.19 Storage protection .....	39

1.19.1	Storage key	39
1.19.2	PSW key field	40
1.19.3	Storage protection logic	41
1.20	Virtual Storage initial concepts	42
1.21	Segmenting a virtual address	44
1.22	Dynamic address translation (DAT) (I)	45
1.23	Dynamic Address Translation (DAT) (II)	46
1.24	Translating large virtual address	47
1.25	Page faults and page data sets	49
1.26	A 64-bit Address Space	51
1.27	Cross memory	53
1.28	Access register mode (data spaces)	54
1.29	z/Architecture time facilities	55
1.30	Server Time Protocol (STP)	58
1.31	Hardware Configuration Definition (HCD)	59
1.32	Logical channel subsystem (LCSS)	61
1.33	Start Subchannel (SSCH) instruction logic	62
1.34	I/O Interrupt processing	66
1.35	DASD controllers (aka DASD controller)	67
1.36	Device number	70
1.37	Subchannel and subchannel numbers	72
1.38	Unit address and logical control unit	74
1.39	I/O summary	75
<b>Chapter 2. Introducing the IBM z14</b>		<b>77</b>
2.1	Introducing the IBM z14	78
2.2	The z14	78
2.3	z14 Technical Highlights	79
2.4	z14 System Design	84
2.5	Processor unit (PU) instances	85
2.6	z14 Hardware models	86
2.7	z14 Sub-capacity models	87
2.8	z14 frames, drawers, and I/O	89
2.9	Processor Drawers	90
2.10	Single Chip Module (SCM)	91
2.11	Processor Unit (PU)	91
2.12	Memory	92
2.13	Power and Cooling	93
2.13.1	Power	93
2.13.2	Cooling	94
2.14	Upgrades	95
2.14.1	Permanent upgrades	96
2.14.2	Temporary upgrades	97
2.14.3	Concurrent upgrades	97
2.14.4	Customer Initiated Upgrade facility	98
2.14.5	Capacity Backup	99
2.14.6	Capacity for Planned Events (CPE)	99
<b>Chapter 3. IBM Z connectivity</b>		<b>101</b>
3.1	I/O channel overview	101
3.1.1	I/O hardware infrastructure	102
3.2	I/O connectivity features	102
3.2.1	FICON EXPRESS	103

3.2.2 zHyperLink Express . . . . .	103
3.2.3 Open Systems Adapter-Express . . . . .	103
3.2.4 HiperSockets . . . . .	103
3.3 Coupling links . . . . .	104
3.3.1 Coupling Express Long Reach (CE LR) . . . . .	105
3.3.2 Integrated Coupling Adapter: Short Reach (ICA SR) . . . . .	105
3.3.3 Host Channel Adapter3: Optical Long Reach (HCA3-O LR) . . . . .	105
3.3.4 Host Channel Adapter3: Optical (HCA3-O) . . . . .	105
3.3.5 Internal coupling . . . . .	105
3.4 Shared Memory Communications . . . . .	106
3.4.1 SMC-Remote Direct Memory Access (SMC-R) . . . . .	106
3.4.2 SMC-Direct Memory Access (SMC-D) . . . . .	106
3.5 Special-purpose feature support . . . . .	106
3.5.1 Crypto Express features . . . . .	107
3.5.2 Flash Express feature . . . . .	107
3.5.3 zEDC Express feature . . . . .	107
3.6 Channel Subsystem . . . . .	108
3.6.1 Multiple channel subsystems concept . . . . .	108
3.7 CSS configuration management . . . . .	111
3.8 Displaying channel types . . . . .	112
<b>Chapter 4. Virtualization and Logical Partition (PR/SM) concepts . . . . .</b>	<b>113</b>
4.1 Virtualization definitions and properties . . . . .	114
4.2 Virtualization concepts . . . . .	115
4.3 Virtualized physical resources . . . . .	117
4.4 Hypervisor types . . . . .	119
4.5 Hypervisor technologies (I) . . . . .	121
4.6 Hypervisor technologies (II) . . . . .	122
4.7 IBM Hypervisors on IBM Z family . . . . .	123
4.8 z/Virtual Machine (z/VM) . . . . .	124
4.9 z/VM options in HMC . . . . .	126
4.10 KVM for IBM Z and DPM . . . . .	127
4.10.1 Hypervisor terminology . . . . .	127
4.11 CPC in PR/SM mode . . . . .	128
4.12 Logical PU concept . . . . .	129
4.13 Shared and dedicated logical PU example . . . . .	131
4.14 PR/SM dispatching and shared CPs . . . . .	133
4.15 PR/SM Weights . . . . .	135
4.16 PR/SM capped versus uncapped . . . . .	138
4.17 Defined capacity or Soft capping . . . . .	140
4.18 Group Capacity . . . . .	141
<b>Related publications . . . . .</b>	<b>145</b>
IBM Redbooks . . . . .	145
Other publications . . . . .	145
Online resources . . . . .	145
How to get IBM Redbooks . . . . .	146





# 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](http://www.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®	IBM Z®	Resource Link®
C3®	IBM z13®	RMF™
CICS®	IBM z13s®	S/390®
Db2®	IBM z14™	VTAM®
DS8000®	Interconnect®	z/Architecture®
ECKD™	Language Environment®	z/OS®
FICON®	MVS™	z/VM®
GDPS®	Parallel Sysplex®	z/VSE®
Geographically Dispersed Parallel Sysplex™	PowerVM®	z10™
HiperSockets™	PR/SM™	z13®
IBM®	RACF®	z13s®
IBM LinuxONE™	Redbooks®	z9®
	Redbooks (logo)  ®	zEnterprise®

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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 z14™ processor design, IBM Z® connectivity, LPAR concepts and Hardware Configuration Definition (HCD).

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, z14 processor design, IBM Z connectivity, LPAR concepts, and HCD
- ▶ 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.

**Patrick Oughton** joined IBM in 2015 after a career as a Mainframe Systems Programmer for over 30 years. He has worked for various organizations in New Zealand and Australia. He is currently an IBM Z Client Technical Specialist for New Zealand and Australian clients.

**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.

Thanks also to Lydia Parziale and Robert Haimowitz, International Technical Support Organization, Poughkeepsie Center, for their support of this project, and the original authors of this book Lydia Parziale and Patrick Oughton.

## 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](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto: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:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ 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.

# 1.1 z/Architecture historical path

Before we provide a more precise definition of a computer architecture, we present the historical evolution of the mainframe computer architecture (see Figure 1-1).

z/Architecture is the next step in the evolution from System/360 to System/370, to System/370 Extended Architecture (370-XA), to Enterprise Systems Architecture/370 (ESA/370), and to 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.

	System/360	System/370	System/370 XA	ESA/70	ESA/390	z/Architect ure
Address Size	24-bit addressing		24-bit or 31-bit addressing			24-bit, 31-bit, and 64-bit addressing
Address Space Size	16M		2G			16Exa
Other		Virtual storage			Expanded Storage Multiple 31- bit AS	
Backward compatibility with the original S/360 architecture and instruction set						

Figure 1-1 z/Architecture historical path

One of the key concepts of a computer architecture is the number of addresses a running program (application or system) may reference to access instructions and operands.

An address space (AS) is defined as the range of addresses available to a computer program and is like a programmer's map of the virtual storage available for code and data. An address space provides each programmer with access to all of the addresses available through the computer architecture. We may say that an AS is the set of all of these addresses.

The number of addresses in an address space depends on the length of the address field where it is stored by the execution processor. This length varies with the computer architecture.

This was a major reason for the introduction of the System/370 Extended Architecture (370-XA) in early 1980, and its operating system, MVS/XA. Because the effective length of an address field expanded from 24 bits to 31 bits, the number of possible addresses in the AS expanded from 16 MB addresses to 2 GB of addresses. The MVS/XA address space is 128 times bigger than the MVS/370 AS.

To maintain compatibility with old application code, IBM z/OS provided two addressing modes (AMODE). Programs that run in:

- ▶ AMODE 24 can use only the first 16 MB addresses.



- ▶ AMODE 31 can use all the 2 GB of addresses.

Similarly, when the address field length grew from 31 to 64 bits in z/Architecture, the maximum possible number of addresses in the address space went from 2 GB, to 16 exabytes (EB), or 8 billion times larger than the former 31 bit. To maintain compatibility with old application code, the IBM z/OS (aka MVS™) operating system provided three addressing modes<sup>1</sup> for programs that run in:

- ▶ AMODE 24 can use only all the first 16 MB addresses.
- ▶ AMODE 31 can use only all the 2 GB of addresses.
- ▶ AMODE 64 can use all the 16 EB addresses.

z/OS and the IBM Z family of hardware products deliver the 64-bit architecture (z/Architecture) to provide qualities of service that are critical to the business world. 64-bit CPC memory support helps eliminate paging. 64-bit CPC memory support may allow you to consolidate your current systems into fewer logical partitions (LP), or to a single native image.

Here, we use the expression “CPC memory” for central storage or real storage or real memory or Random Access Memory (RAM).

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

## 1.2 Computer architecture overview

This chapter introduces at a medium to high level, the concepts of the z/Architecture. It is a massive simplification of the *z/Architecture Principle of Operations, SA22-7832 (POP)* with its more-than-one-thousand pages.

Now, a key question. Should you skip or read this chapter? The answer is simple. Are you certain that your daily production workload will be processed smoothly, with no problems with these disciplines: availability, performance, integrity, security, and compatibility? If your answer is yes, skip this chapter and go directly to Chapter 2, “Introducing the IBM z14” on page 77.

In this section, you find the basics. These basics matter when you do have problems.

On top of that, the writers of this chapter are not aware of any IBM publication that makes the POP more easy to understand.

### Computer architecture

This chapter provides a detailed description of z/Architecture, and it discusses aspects of the computers (CPCs) running such architecture. These CPCs are now referred to as IBM Z and they comprise the: IBM z800, z890, z900, z990, z9® EC and BC, z10™ EC and BC, z196 EC and BC, zEC12, zBC12, IBM z13®, IBM z13s® and z14.

So, what is a “computer architecture”? The computer architecture of a computing system defines its attributes as seen by the programs (software) that are executed in this system, that is, the conceptual structure and functional behavior of the computer hardware (see Figure 1-2). 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 internal organization of the data flow, the

---

<sup>1</sup> For more information on AMODE, see section 1.16, “Addressing mode” on page 30

logical design, the physical design, and the performance of any particular hardware implementation.

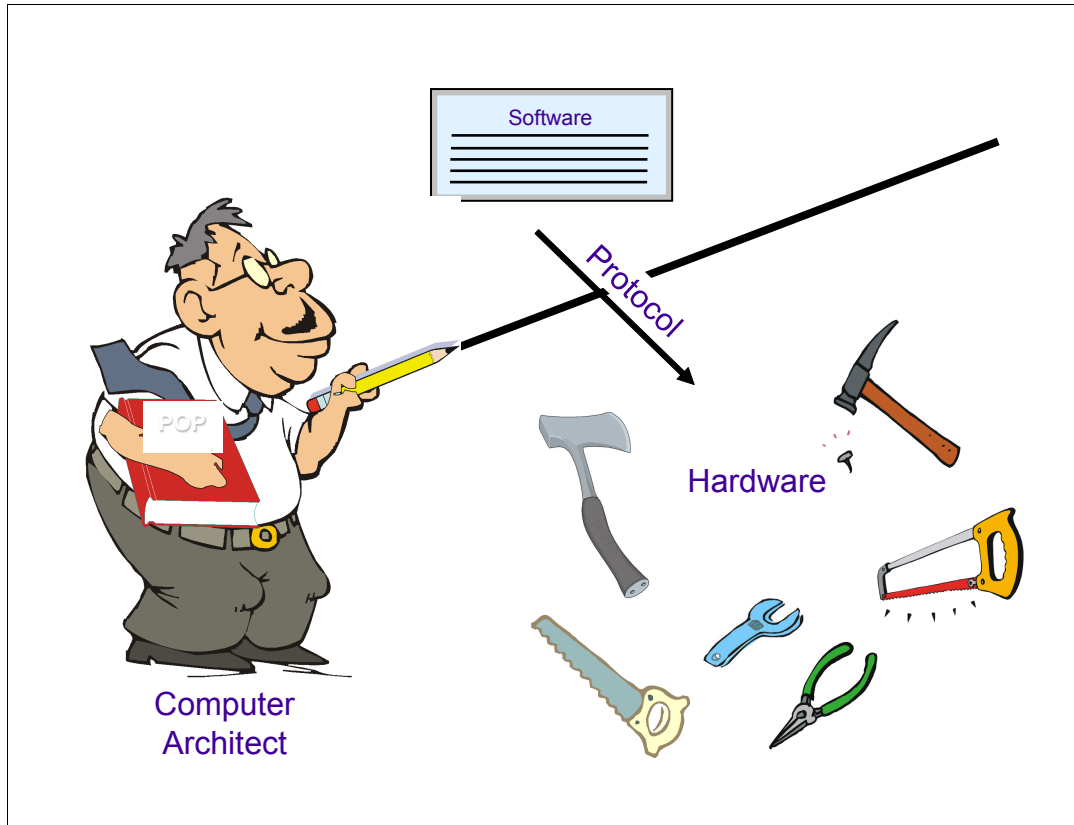


Figure 1-2 An architecture definition

Several dissimilar computer implementations may conform to a single architecture. This happens when the execution of a set of programs on different computer implementations produces the results that are defined by a single architecture, the implementations are considered to be compatible for those programs.

In this chapter, we describe the z/Architecture, that is composed of the following key elements:

- Set of instructions (more than 1000 different ones)
- Program status word (PSW) and interruptions
- Accessing CPC memory and registers
- Execute an I/O operation
- Implement memory Protection mechanism
- Cross Memory
- Virtual memory

**Note:** When reading this chapter, it is useful to have the *z/Architecture Reference Summary*, SA22-7871 on 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.

## 1.3 Concept of a process

There are many concepts in z/Architecture, such as multiprocessing, that in order to be understood you need to be familiar with the academic concept of a process. In mainframe terminology, a process is called a dispatchable unit (DU) or unit of work.

As defined in a commercial data processing environment, a process is the serial execution of programs (and its instructions) in order to solve one problem of a business unit, such as, payroll update, banking checking account transaction, Internet query, and so on.

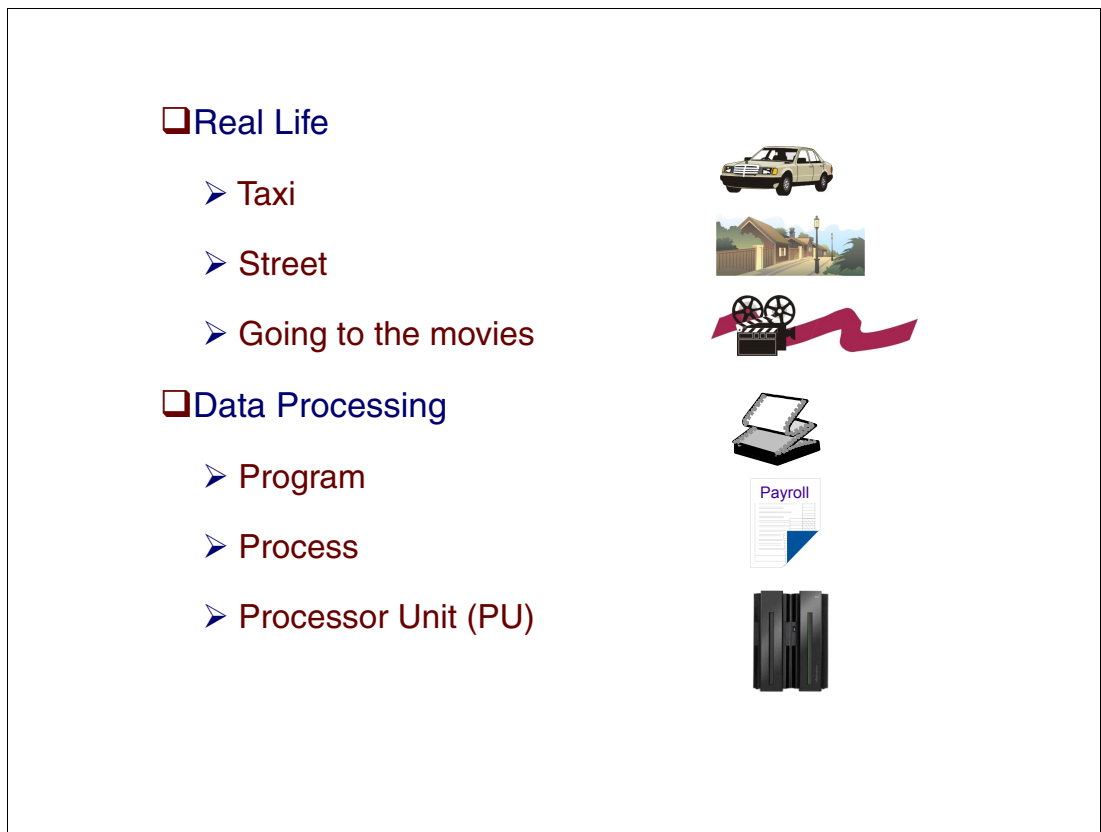


Figure 1-3 Concept of a process

### Analogies

To illustrate the process concept, let 's draw an analogy using real-life terms.

Each item in the list labeled "Real Life" in Figure 1-3 corresponds to an item in the Data Processing list: the taxi is the Processor Unit (PU) or CP, the street is the program, and "going to the movies" is the process.

Streets and taxis were created because people need to go to places. Programs and PUs (and thus, the data processing industry) were developed because processes need to be executed on behalf of business transactions.

So, if every time you go to the movies, you notice that you are late because you have a slow taxi, it would be useless to call another slow taxi, the same way, if your process is taking longer because of a slow PU, it is useless to add another PU that has the same speed.

You could also go to the movies by taking a taxi, then get out somewhere, do some shopping then take another taxi to continue on to the movies, and so on.

Similarly, the same process can start in one PU, be interrupted (for example, by an I/O interrupt), and later resume in another PU, 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 PUs on behalf of different processes.

Taking someone to the hospital in an ambulance allows you to shortcut the traffic queue, just as a key process has a higher priority in a queue for getting PU and I/O compared with other processes.

A process may create another process to execute both in parallel. For example, the parent process "going to the movies" may attach another daughter process "buy a cake for Grandma". It implies two taxis taking different people through streets or different processes executing programs (the same or not) in different PUs.

Putting it together, a program is a set of instruction whereas process is program in execution. Program is passive entity whereas process is active entity. Program has single instance whereas process has several instances.

## States of processing

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

**Active:** In the Active state, a process program is being executed by one PU.

**Ready:** In the Ready state, a process is delayed because all available PUs are busy executing other processes.

**Wait:** In the Wait state, a process is being delayed for a reason that is not PU-related, for example, waiting for an event, such as an I/O operation to complete.

## Process attributes

In the operating system, processes are usually born depending on the needs of business transactions introduced in the system through the network.

A process dies normally when its last program completes (i.e., its last instruction gets executed) without any error. 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 (PU cycles, number of I/O operations, amount of consumed memory) 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 how important the process is, and not on the program.

Then, for each non-active process in the system, z/OS must keep the following information in process control blocks, such as the task control block (TCB) and the service request block (SRB):

- ▶ State (Program Status Word (PSW) and registers contents)
- ▶ Resources held (data sets opened, virtual memory getmained, programs loaded in memory, owned ENQ resources due to integrity).
- ▶ Priority when entering in queues for resources
- ▶ Accounting - how much PU time was used, I/Os executed, and memory occupied.
- ▶ Addressing - which addresses the programs of the process have the right to access in CPC memory (for data and instructions).
- ▶ Security profile - which data resources are allowed to be accessed by the process programs.

### Dispatchable units (DU)

In z/OS, processes are called dispatchable unit (DU), which consist of tasks (represented by a TCB control block) and service requests (represented by an SRB control block). 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). An address space must be designed to contain the addresses referred by the executing program of the new DU. Refer to “Addresses and address spaces” on page 28.

An address space has naturally several programs from several DUs, but just one (the step task) usually and optionally creates other DUs. The others are naturally in wait state.

The benefits of multiprocessing (several live DUs active in parallel in the same system) can usually be achieved with only several step tasks of several address spaces.

**Note: From now on in this publication, we will use the acronym DU instead of "process".**

## 1.4 System components

Before giving more detailed information about z/Architecture, we will describe a system generically (see Figure 1-4).

A *system* is made up of hardware components, including a processor unit (PU), and software products, with the primary software being an operating system such as z/OS. Other types of software (system and user application programs) also run on the system. The processor unit (PU) is the functional hardware unit that interprets and processes program instructions. The PU and other system hardware, such as channels and memory, make up a server complex (CPC).

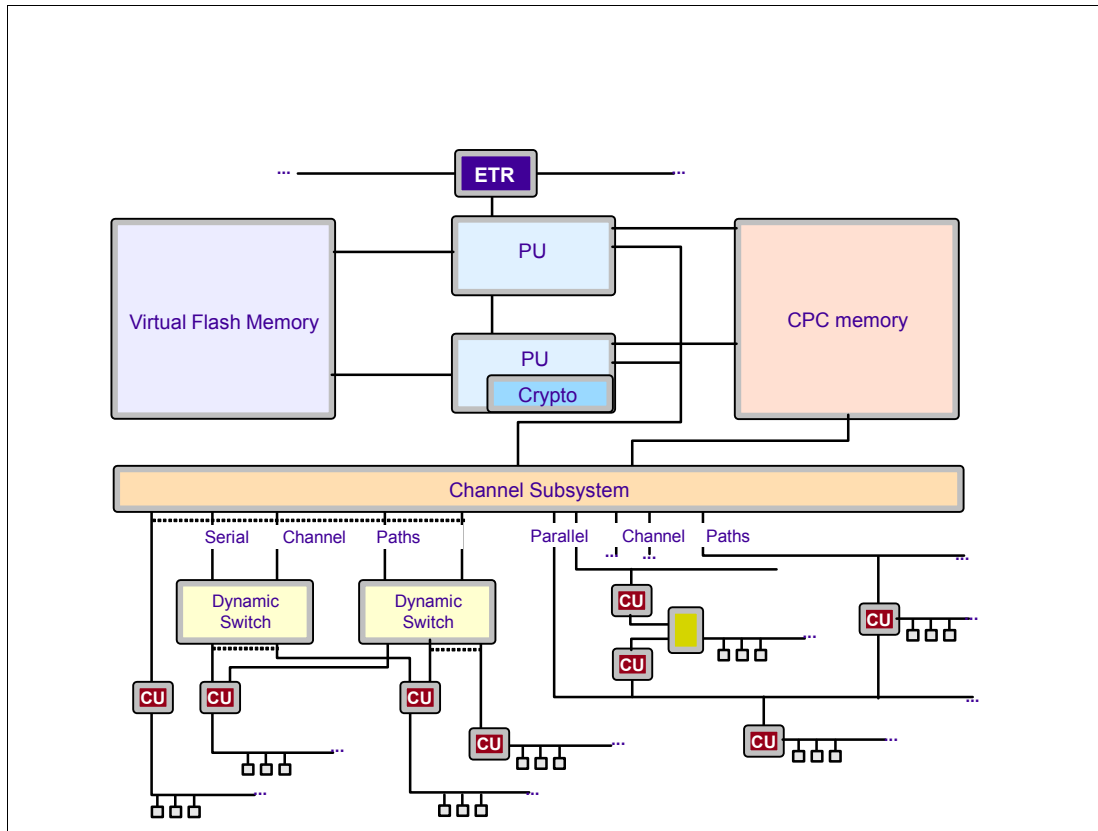


Figure 1-4 System components

Physically, a system (computer) consists of the following:

- ▶ CPC memory to keep data and programs. In a z14, its size is up to 32 TB. However, z/OS supports only 4 TB in its logical partition.
- ▶ Processor Units (PU)-with different personalities selected by the customer, such as: CP, IFL, zIIP, ICF, IFP, SAP and Spare. The CPC model number designates the maximum number of processor units (PUs) available for an installation to use.
- ▶ Hardware management facilities (Support Element and Hardware Management Console (HMC), which is not represented in Figure 1-4).
- ▶ A channel subsystem formed by system assisted processors (SAPs) and I/O channels (IBM FICON®, zHPF and PCIe HyperLink).
- ▶ I/O devices (for example, disks also called DASD, tape, printers, teleprocessing); I/O devices are attached to the channel subsystem through controllers. The connection between the channel subsystem and a controller is called a channel path. I/O devices contains data and programs that are stored in a large and non-volatile storage.

### 1.4.1 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.

## 1.4.2 Channels

A *channel* is a processor - much simpler than a PU - in charge of executing an I/O operation. An I/O operation is a dialog between two processors through a fiber optic cable. One is a channel located at the PCIe I/O Drawer of a z14 and the other, is a processor named Host Adapter (HA) located in an I/O controller, such as a DS8880, where your data and programs are securely stored. The reason for the dialog is to move such data and programs between the z14 CPC memory (located at the CPC Drawer) and the I/O controller. In a sense, the PU offloads to the channel the work associated with I/O operation execution.

Because the commercial workload is prone to I/Os (I/O bound), a z14 may allow hundreds of channels. There are three types of channels at the z14 CPC, depending mainly on the used protocol along the dialog:

- ▶ IBM FICON Express 6S+.
- ▶ IBM Z High Performance FICON (zHPF), that is similar to FICON but more efficient.
- ▶ PCIe HyperLink, a z14 exclusive, more modern and five times higher I/O rate than zHPF.
- ▶ CF links to access the Coupling Facilities. Refer to the Figure on page 76.

## 1.4.3 Dynamic Switch

The Dynamic Switch is a piece of hardware that allows a more flexible and less error-prone communication between channels and controllers. It is capable of providing multiple connections between entities connected to the ports of the switch (that is, between channels and I/O control units).

## 1.4.4 IBM Virtual Flash Memory (VFM)

Virtual Flash Memory is the next generation of storage class memory designed to help improve availability and performance during workload transitions for improved quality of service. Virtual Flash Memory can help reduce latency for critical paging that might otherwise impact the availability and performance of your key workloads.

VFM is a part of the CPC memory (up to 10 TB) not directly accessed by the PU. z/OS keeps track of the VFM objects such as: page data sets, large pages, overflow area of coupling facility structures, SVC dumps and so on. At zEC12 CPC, the Flash memory cards were located in slots of the PCIe I/O Drawer.

## 1.4.5 Pervasive encryption

Cryptography must be pervasive nowadays to protect from security breaches. Pervasive encryption is a consumable approach to enable extensive encryption of data in-flight and at-rest to substantially simplify encryption and reduce costs associated with protecting data and achieving compliance mandates.

The IBM Z platform is designed to provide pervasive encryption capabilities to help you protect data efficiently in the digital enterprise.

IBM z/OS was designed to take advantage of the z14 platform embedding the use of the z14 cryptographic engines within the operating environment. This approach helps to create an environment where policies can be enforced that govern intrinsic data protection, helping you build a perimeter around business data.

To speed up cryptographic computing and to save PU cycles, the z14 offers two hardware solutions:

- ▶ CPACF a cryptographic processor inside of the PU processor. It offers a set of symmetric cryptographic algorithms of clear key operations for SSL, VPN, SHA, RSA and data-storing applications. Starting at z14, the CPACF is able to implement non-clear key algorithms.
- ▶ Crypto Express 6S card located at the PCIe I/O Drawer. It supports non-clear key cryptographic algorithms such as: AES, DES based symmetric and the Public Key Algorithm (PKA) asymmetric algorithms. Due to the distance to the PU, this hardware solution is slower than the CPACF, but faster than a cryptography software solution.

## 1.5 Processing units (PUs)

Figure 1-5, shows some of the PU types (or personalities) that can be ordered (enabled or assigned) on a z14 CPC for its different logical partitions.

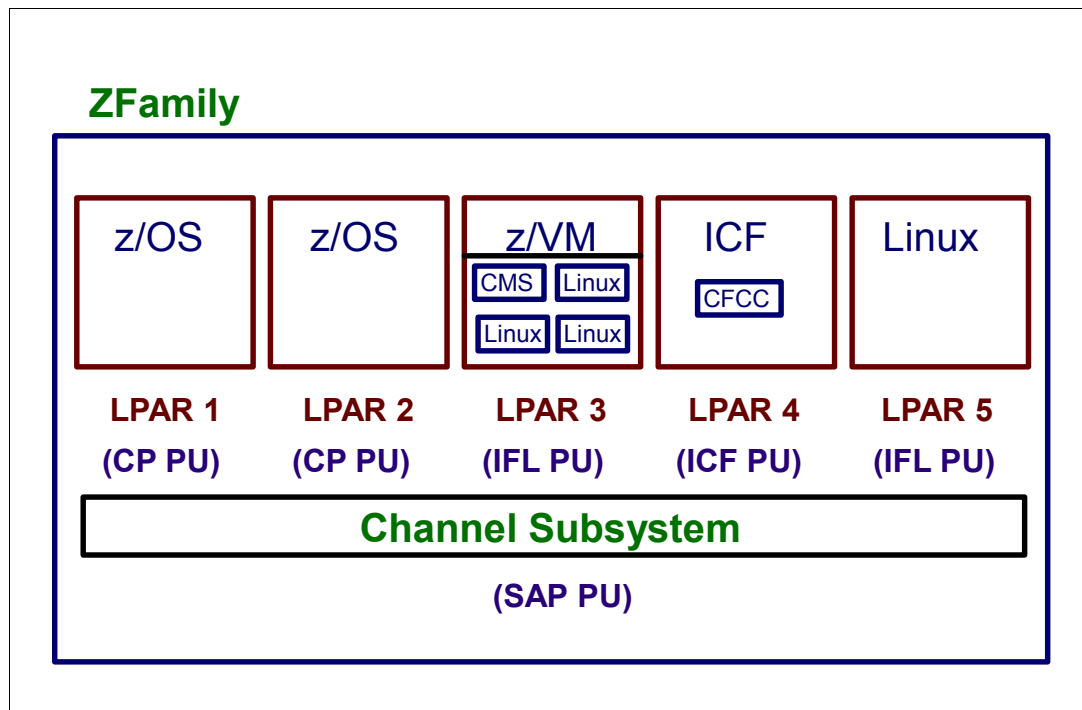


Figure 1-5 Logical Partitions and PU types

- CP** A CP is a general purpose processor that is able to execute all the possible z14 running operating systems, such as z/OS, zLinux, z/ VM, IBM z/VSE®, Coupling Facility Control Code (CFCC), and z/TPF. A CP is also known as a CPU. All other PU types are less expensive than PUs.
- IFL** This type of PU is only able to execute native zLinux and zLinux under IBM z/VM®. IFLs are less expensive than the CPs.
- 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 the CPs.
- zIIP** This type of PU is to run in z/OS only, for eligible IBM 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 CP 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. The choice of this percentage and of the products with zIIP affinity belongs to IBM.

- IFP** A single PU dedicated solely for the purpose of supporting the native PCIe features (10GbE RoCE, zEDC Express and HyperLink Express) and it is initialized at POR if these features are present. Unlike other characterized PUs, it is free of charge.
- SAP** A System Assist Processor (SAP) is a PU that manages the starting and the end of an I/O operation required by the operating systems running in all CPC logical partitions. It frees z/OS (and the PU) from this role, and is “mainframe-unique”. SAP also is in charge of executing the Service Time Protocol (STP) that synchronizes all the CPC TOD clocks. z14 models have a variable number of standard SAPs configured. Refer to “Server Time Protocol (STP)” on page 58.
- Spare** A spare PU is a PU that is able to replace, automatically and transparently, *any* failing PU in the CPC (very seldom). Also, it can assume any PU personality in a situation of increasing demand. There are at least two spares per z14, but usually you have much more, as a reserved capacity.

Understand that all the above described PUs are identical from a hardware point-of-view.

## 1.6 Microcode Concepts

To better explain how some z/Architecture instructions are implemented on the z14, we introduce the concept of *microcode*. Some of the z/Architecture instruction set is implemented through microcode. Microcode is a design option (not an architecture decision) that is used to implement the PU instruction logic. To make it simple, we can divide the PU into two pieces: *data control* and *data flow* (also known as the arithmetic logic unit or ALU). Instructions are really executed in the data flow (where data is transformed); however, the sequence and timing of each of the multiple suboperations done in the data flow is ordered from the data control. As shown in Figure 1-6 on page 12, 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 play (execute) properly.

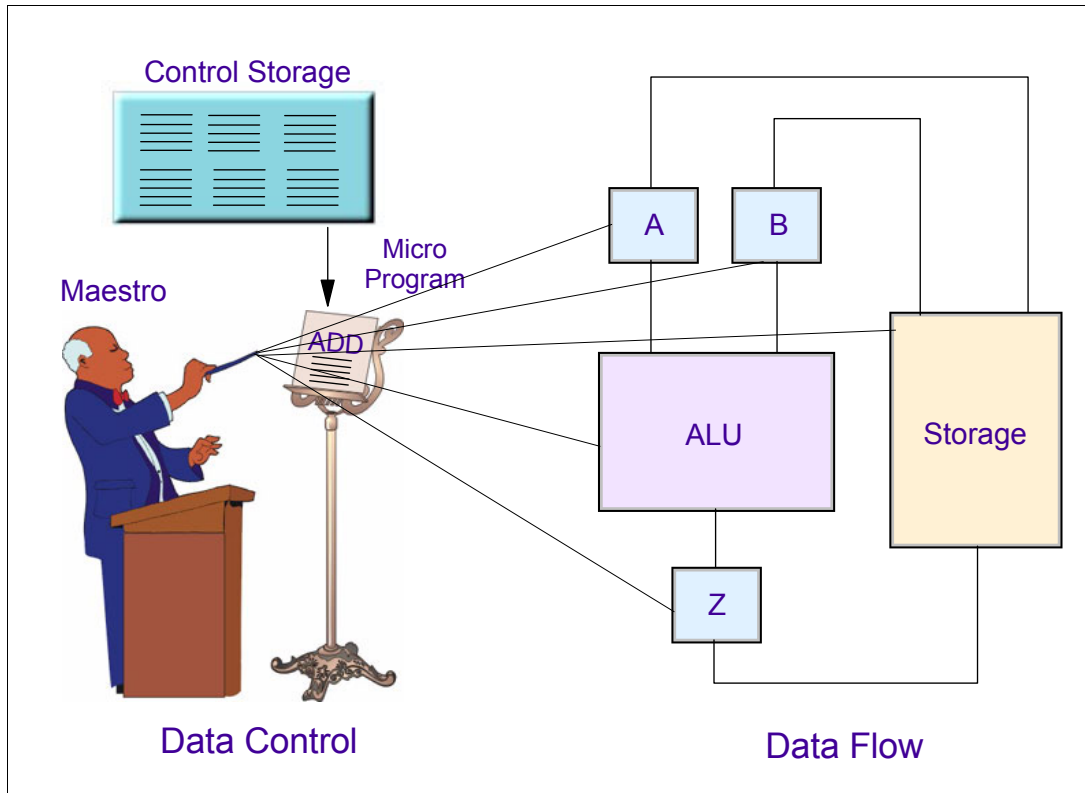


Figure 1-6 Microcode analogy

In a microcoded PU instruction, 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 time sequence (including parallelism) of orders to be sent by the data flow. These micro programs are loaded in a special internal memory in the PU, called *control storage*, at power-on reset (POR) time, by the Support Element.

Decoding a microcoded instruction consists of finding the address in the control storage of its pre-loaded 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 even a new instruction (creating a new function) can be implemented by just changing or adding to the existent microcode. It is also possible that the same PU 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 memory. The advantage of hardwiring is performance because there is no data control overhead. In z14, due to these performance reasons, a large majority of z14 instructions are hardwired.

## 1.7 z/Architecture Components

The z/Architecture major components are:

- Registers
- Instruction set
- Current Program Status Word (PSW)
- Multiprocessing

- Data formats
- Prefix Storage Area (PSA)
- Initial Program Load (IPL)
- Interrupts
- Storage protection
- Addressing memory locations (virtual memory)
- PU signaling facility
- Time measurements and synchronization
- I/O operations
- Coupling Facility

## 1.8 PU Registers

PU registers (Figure 1-7 on page 13) are a fast memory within a PU, like the current PSW. The PU provides registers that are available to programs, but that do not have addressable representations like bytes in CPC memory. There are several types of registers, as explained in the following sections: general purpose registers (GPR), floating-point registers, control registers, access registers, and prefix register. Each type (with the prefix exception) has 16 registers per PU of 8 bytes each (but access registers have only 4 bytes).

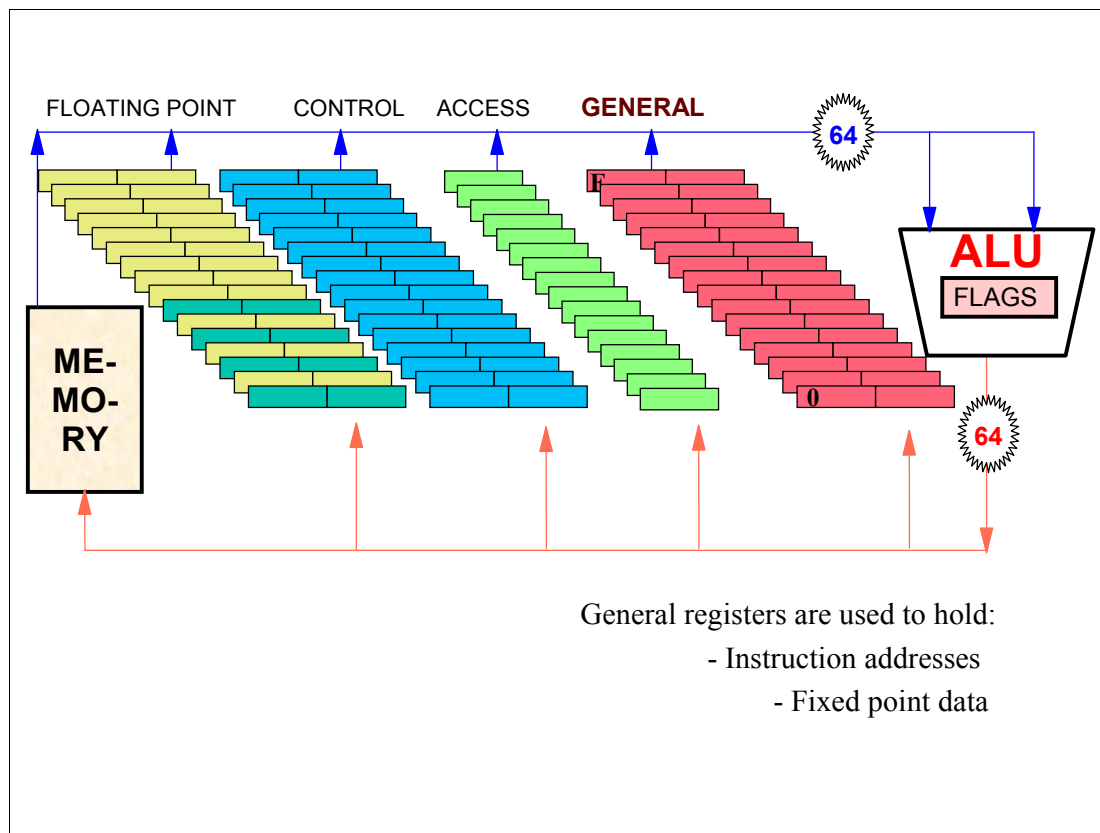


Figure 1-7 PU registers (general)

Simultaneous Multi Thread (SMT) of the processor cores - the zIIPs and SAPs in the z14 - means that several tasks can run concurrently in the same PU (each one in its own thread ID acting effectively like a logical CPU having its own set of related management control blocks and associated metrics).

The instruction operation code determines which type of register is to be used in the operation associated with this instruction.

### 1.8.1 General purpose registers (GPR)

GPRs are used by Assembler instructions:

- ▶ As a base-address register or index register, to inform to the executing PU, a memory operand address.
- ▶ To keep temporary data (operands) loaded from memory to be processed within such a register or already processed. As an example, they can be used as accumulators in general arithmetic and logical operations.

GPRs are identified by the numbers 0-15 (register numbers 00 through 0F), and are designated by a four-bit R field in an instruction. Usually, when a GPR contains data, this data is in binary integer format, also called *fixed point*. There are certain PU instructions that are able to process data loaded and stored in GPRs. Load-type instructions copy data from the CPC memory to GPRs, and Store-type instructions copy data from GPR to memory.

When an instruction needs only 32 bits, such as Load, Add, and Store, only bits 32 to 63 of the GPR are used, the high order bits 0 to 31 are not used.

z/Architecture provides instructions that use 64-bit operands to produce a 64-bit binary integer. These instructions include a “G” in the instruction mnemonic, Load (LG), Add (AG) and Store (STG) that handle all 64 bits of the GPR. Those instructions that have 64- and 32-bit operands have a “GF” in the mnemonic such as Load (LGF) and Add (AGF).

#### Control registers (CR)

CRs are used, as a communication area between the operating system (z/OS) and the z14 hardware z/Architecture. Then, the bit positions of some of the CRs are assigned to particular hardware facility, and are used either to specify that an operation can take place, or to furnish special information required by the facility.

Summarizing, 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 memory, and clocks) implemented in the PU. It is a kind of extension of the PSW (covered in “Program status word (PSW)” on page 17. Refer to *z/Architecture Reference Summary, SA22-7871*, for the complete set of CR contents.

The CRs 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.

#### Access registers (AR)

ARs are used mainly by z/OS to access *data spaces*, through activating the access register mode in the PU via PSW bits; this subject is covered in more detail in “Access register mode (data spaces)” on page 54. To make it simple, an AS has virtual addresses of instructions and its operands (data) and a data space has the virtual addresses of operands only. Data space implements an idea from the universal, that is, to separate programs from their data.

An *access register* consists of 32 bit positions containing an indirect specification of a segment table. This table is used by the dynamic-address-translation (DAT) hardware mechanism (within the PU) to translate virtual addresses into real addresses.

When the PU is in a mode called the *access-register mode* (controlled by bits 16 and 17 in the PSW), an instruction B field (in addition to its role of specifying a virtual address for a memory-operand reference) designates an AR that points indirectly to the data space segment table to be used by DAT to translate this virtual address.

Instructions are provided (such LAM) for loading and storing the contents of the access registers, and for moving the contents of one access register to another.

## 1.8.2 Floating point registers

All floating-point instructions (HFP, BFP, and DFP) use the same set of floating-point registers (FPRs). The FPRs are identified by the numbers 0-15 and are designated by a four-bit R field in floating-point instructions. Each FPR is 64 bits long and can contain either a short (32-bit) or a long (64-bit) floating-point operand. This floating-point data type includes with the number, its signal and fraction point. Refer to “Floating-point numbers” on page 26.

As shown in Figure 1-8, pairs of FPRs 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.

Then, FPRs are used to keep temporary data (operands) loaded from CPC memory to be processed. This data must be in the format HFP, BFP or DFP.

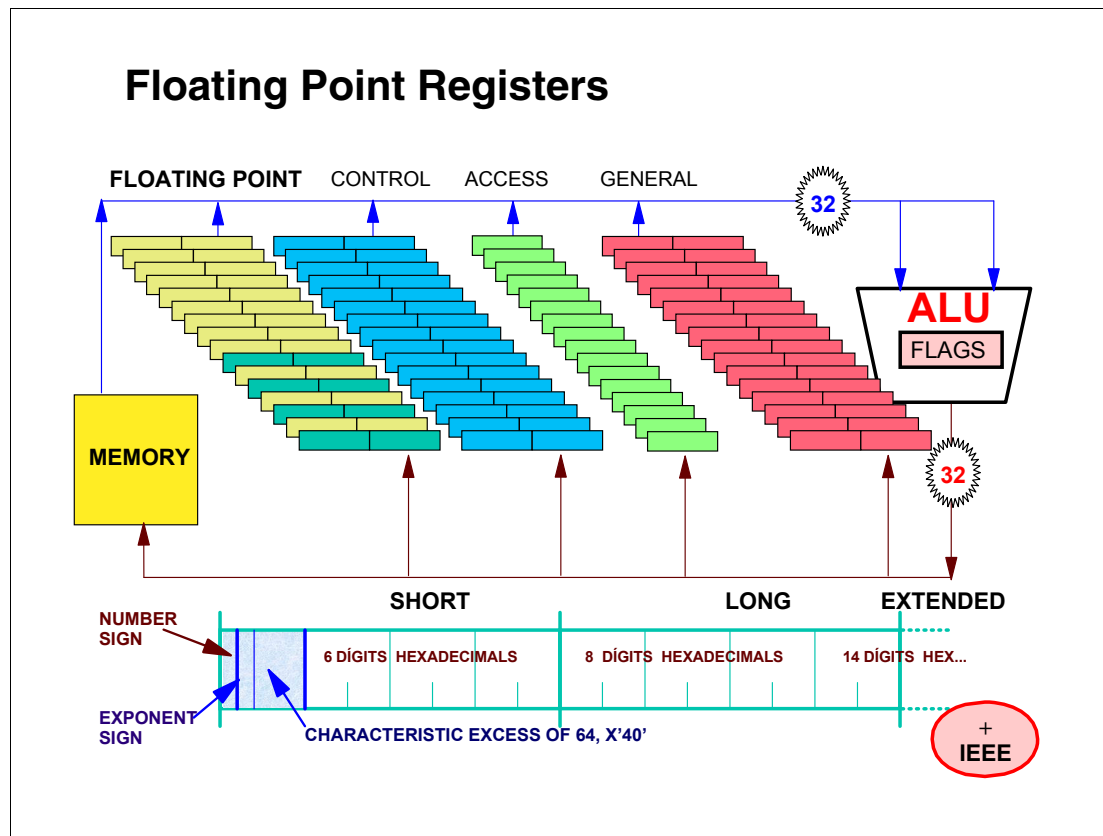


Figure 1-8 Floating point registers

## 1.9 Instruction set and its formats

The *instruction set* is a key item of the z/Architecture, and it provides the programmers access to the hardware functions through the use of written programs (a logical set of instructions). In other words, the PU 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 z14 PUs and to address different kinds of problems, mainly the ones of commercial workloads.

There are more than 1,300 instructions, as defined in z/Architecture and the majority of them have their implicit logic described in *z/Architecture Principles of Operations*, SA22-7832.

Each instruction set for z/Architecture has many instructions (compared with other platforms), because IBM Z is a Complex Instruction Set of Computing (CISC) machines, as required by commercial data processing. Many of these instructions operate on 64-bit binary integers and also address memory operands with 64-bit. For compatibility reasons, the instructions of the previous mainframe architectures are carried forward to the z/Architecture. Each new family of IBM Z PUs adds more complex and more useful instructions into the z/Architecture. The major objective of these instructions is to decrease the total number of instructions executed (also known as the Path Length), and as a result, to decrease your PU time. It is recommended that you recompile your more frequently used code with a modern compiler in order to exploit these new and more powerful instructions.

An example of compatibility between architectures is that in an ESA/390, the general purpose register (GPR) has 4 bytes (32 bits) and z/Architecture is double the size. The bit positions of the GPR of z/Architecture are numbered 0-63. An ESA/390 instruction that operates on bit positions 0-31 of a 32-bit GPR 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.

### Instructions

An *instruction* is one, two, or three halfwords (two bytes) in length, as shown in Figure 1-9, and must be located in storage on a halfword boundary. Its size is dependent on how much information needs to be passed to the PU.

Each instruction consists of two major parts:

- Op codes** The Operation codes, or instruction codes, indicate to the PU which instruction (or operation) to perform. For example, ADD (A) has an Op code of 5A; CHECKSUM (CKSM) has an Op code of B241. Also, by the contents of the first two bits, there is an indication about the instruction length.
- Operands** The operands are the remainder of the instruction, following the Op code. They can be in CPC memory (the instruction indicates the address), or in a register contents (the instruction indicates the register number).

All instructions that process operands in CPC memory need to address that operand, usually through a virtual address. For that, the PU adds the contents of the following:

- ▶ Contents of a GPR indicated in the instruction as a base, such as B1 (as shown in Figure 1-9 on page 17) or an index, such as X2 (as shown in Figure 1-9 on page 17.)
- ▶ A displacement is indicated in the instruction, such as D1 (as shown in Figure 1-9 on page 17). For the RSY, RXY and SYI type of instruction, this displacement is 20 bits (1 MB range); for the others, it has 12 bits.

## Instruction formats

In RISC architecture, all the instructions have the same length, usually 4-bytes. In z/Architecture, each instruction is in one of 26 formats.

Figure 1-9 provides an example of some of the instruction formats available.

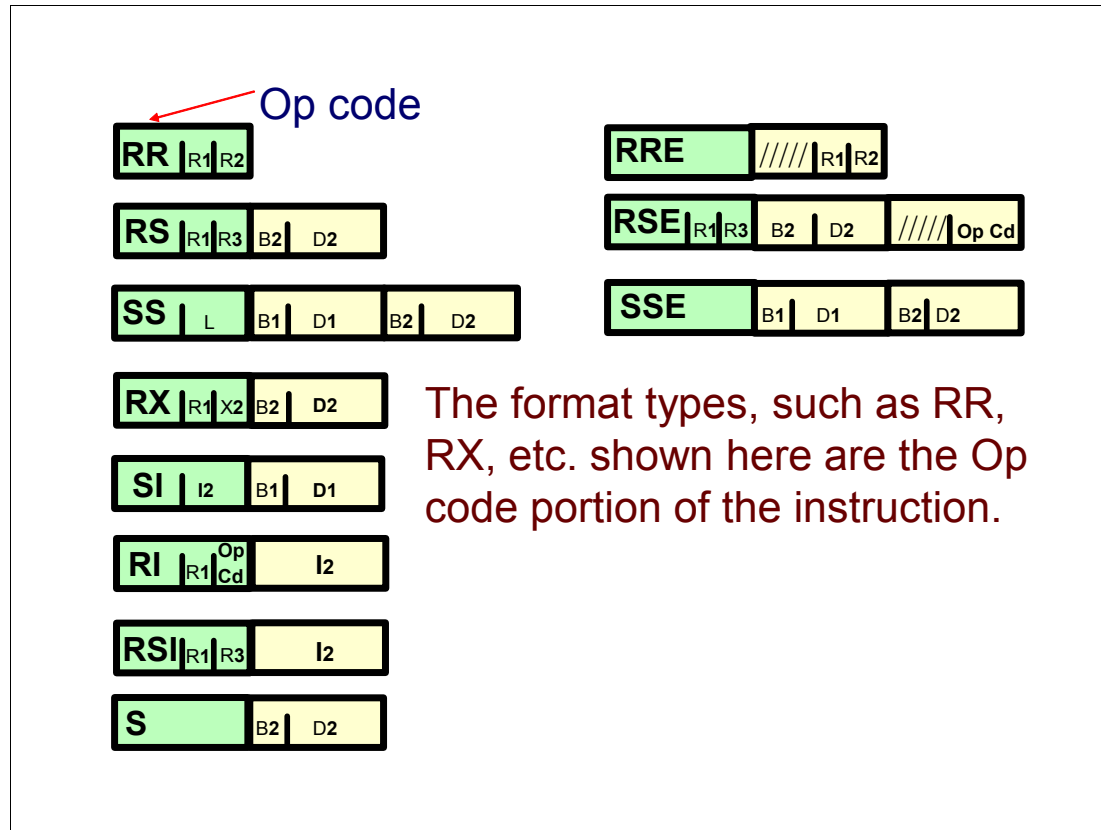


Figure 1-9 Several instruction formats

The reason for so many formats is the diversity of the information to be passed to the PU by the programmer. As an analogy, if you need to go to a nearby grocery store, you might use a bicycle (a 2-byte RR format of instruction); but if you go to a place that is farther away, you might drive a car instead (a 6-byte SSE format of instruction) to save time.

For more information and a description of the various instruction formats, see the *z/Architecture Principles of Operation, SA22-7832* which can be found at:

<http://www-01.ibm.com/support/docview.wss?uid=isg2b9de5f05a9d57819852571c500428f9a>

## 1.10 Program status word (PSW)

The current PSW is a memory circuit located within each PU and contains information required for the execution of the currently active program in the active task. That is, it contains the current state of the PU. It has 16 bytes or 128 bits (shown in Figure 1-10).

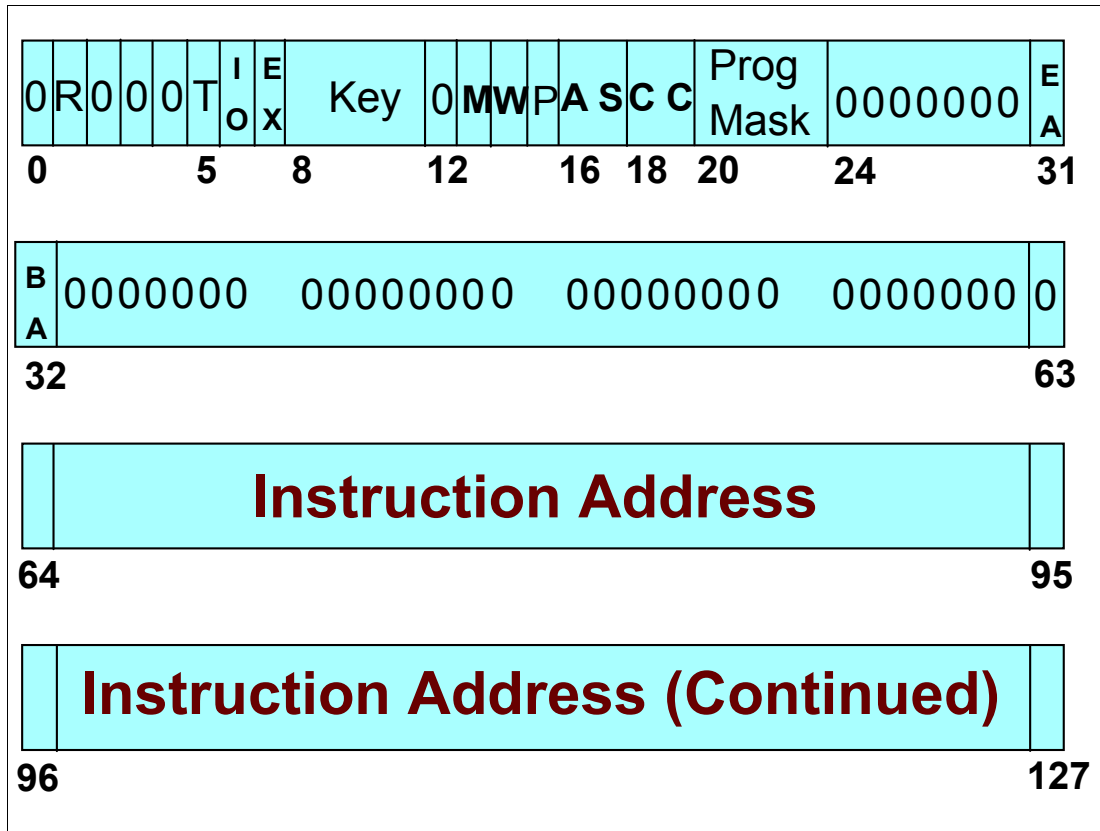


Figure 1-10 The current PSW general layout

The PSW includes a system mask, the PSW key, any problem state, the instruction address, condition code, and other information used to control instruction sequencing and to hold and indicates the status of the PU. In other words, the *current PSW* governs the program currently being executed.

With Simultaneous Multi Thread (SMT) at the zIIPs and SAPs in z14, that means, several tasks running can run concurrently in the same PU (each one using a thread ID). In this case, there is one current PSW per thread ID in the same PU.

### 1.10.1 PSW format

In this section, we describe the current PSW format. Figure 1-11 provides a graphical view of some of this detail.



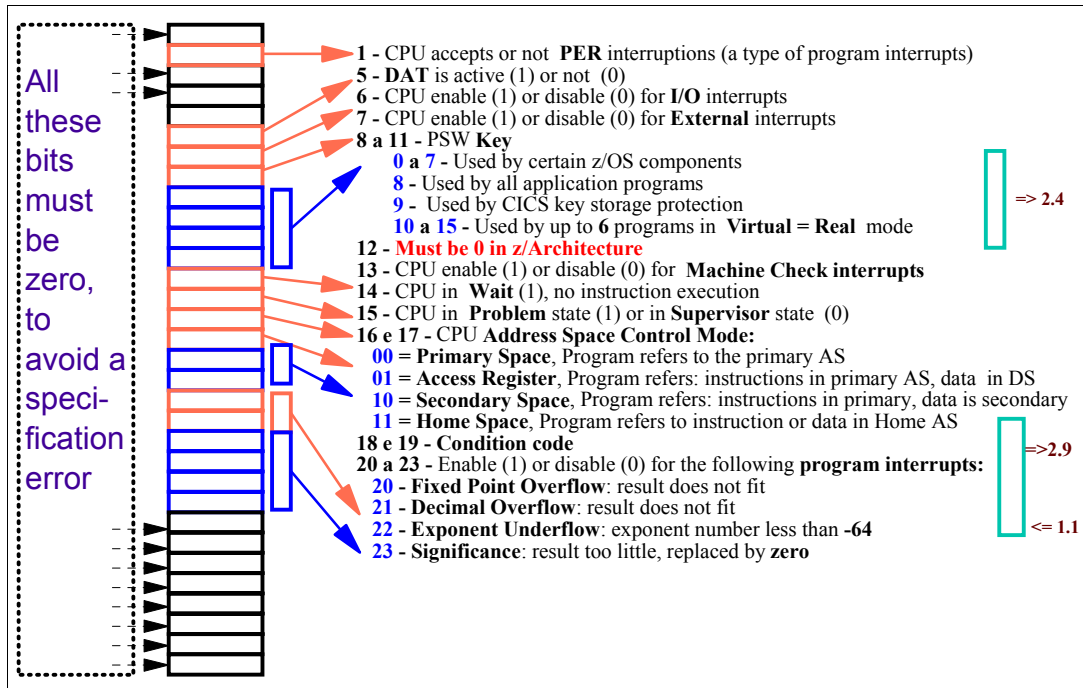


Figure 1-11 PSW Bits detail description

Bits 0-7 of the PSW are collectively referred to as the system mask. A summary of the functions can be found in the *z/Architecture Principles of Operation*, SA22-7832 which can be found at:

<http://www-01.ibm.com/support/docview.wss?uid=isg2b9de5f05a9d57819852571c500428f9a>

### PER mask

Shown as an R in Figure 1-10, this is the program event recording mask. Bit 1 (shown as number 1 in Figure 1-11) controls whether the PU is enabled for interrupts associated with program-event recording. 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. PER is a hardware z/Architecture function to assist in debugging z/OS code.

### Dynamic Address Translation (DAT) mode

Shown as a T in Figure 1-10, Bit 5 controls whether implicit DAT translation of virtual address of instructions and operands located in CPC memory. When the bit is zero (0), DAT is off, and CPC memory addresses are treated as real addresses. When the bit is one (1), DAT is on, and the DAT PU mechanism is invoked. Only a few z/OS routines run with DAT off. Also, the coupling facility control code (CFCC) operating system (which runs in the coupling facility logical partition) runs totally with DAT mode bit 5 off.

### I/O mask

Shown as IO in Figure 1-10, Bit 6 controls whether the PU is enabled for I/O interruptions. An I/O interruption is demanded by a channel when an I/O operation is over.

When the bit is zero, an I/O interruption cannot occur in this PU. 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. Usually in z/OS, these CR6 bits are on.

There are certain z/OS programs that cannot be interrupted, to avoid raising data integrity exposures in CPC memory. It is also important to know that the incoming I/O interrupt is not lost when all the PUs are disabled for I/O. They are kept at SAP hardware queue.

At z14 using the PCIe HyperLink for 4-KB random read hits, there are not I/O interrupts, because in such case the I/O operation is very fast (tens of microseconds) and it is synchronous, that is, the PU loops, thus testing the end of the I/O operation.

### External mask

Shown as EX in Figure 1-10, Bit 7 controls whether the PU is enabled for interruption by conditions included in the external type of interrupt. 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. Usually in z/OS, these CR6 bits are on.

There are certain z/OS programs that cannot be interrupted, to avoid rising data integrity exposures in CPC memory. It is also important to know that the incoming external interrupt is not lost when all the PUs are disabled for external interrupts. They are kept at SAP hardware queue.

**Note:** An incoming external interrupt is not lost when all the PUs are disabled for external interrupts. It is kept in the SAP hardware queue.

### PSW

Indicated by the word Key in Figure 1-10 and positioned in bits 8-11, the PSW key is used by a hardware mechanism within the PU called *memory protection*. It guarantees that programs running DUs do not alter or read areas in CPC memory that belong to other DUs; refer to “Storage protection” on page 39 for more information.

### Architecture Mode

This bit 12 must be off (0) and indicates 128-bit z/Architecture mode, which is the only possible architecture to execute in a z14.

### Machine-check mask

Indicated by an M in Figure 1-10, bit 13 controls whether the PU 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.

This bit is off when the z/OS routine, Machine Check Handler (MCH), is trying to recover from a hardware machine check and recursively receives another machine check interrupt.

### Wait state

Indicated by a W in Figure 1-10, when bit 14 is on, the PU is waiting; that is, no instructions are processed by the PU, but interruptions may take place. When bit 14 is zero, instruction fetching and execution occur in the normal manner. When in wait state, the only way of getting out of such state is through an Interruption, which is covered in 1.17, “Interrupts” on page 31, or by an IPL (a z/OS “boot”).

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

When z/OS is running under IBM PR/SM™ (as is mandatory in z14) in a shared logical PU, and there is no ready DU to get this logical PU, bit 14 is set on by z/OS itself. In this case, it causes a PR/SM intercept that forces the physical PU to leave the logical PU. Later, when PR/SM cannot find any shared logical PU to get to the physical PU, bit 14 is switched on again. In this case, the physical PU enters into a real wait state.

In performance management, there is a very important metric named physical PU utilization, that is defined by the complement to 100% of the wait state (bit 14 on).

### Problem or supervisor state

Indicated by a P in Figure 1-10, bit 15 is the problem (or supervisor) state.

PU instructions can be classified as *privileged* and *unprivileged*. Note that, if misused, privileged instructions may damage system integrity and security. When the PU is in the supervisor state (bit 15 is 0 or Off), all instructions are valid and can be executed. When the PU is in the problem state (bit 15 is 1 or On), only those instructions that provide meaningful information to the problem program that cannot affect system integrity are valid; such instructions are also called unprivileged instructions.

Privileged instructions should only be executed by reliable z/OS code. 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 beyond the bit 15 state.

When a PU in the problem state attempts to execute a privileged instruction, a privileged-operation program interruption is recognized and executed.

### Address-space control

Indicated by an AS in Figure 1-10, bits 16 and 17, along with PSW bit 5, control the translation mode. If the AS is:

- ▶ 00 - Primary-space mode
- ▶ 01 - Access-register mode
- ▶ 10 - Secondary-space mode
- ▶ 11 - Home-space mode

Refer to “Addressing mode” on page 30, for more information.

### Condition code

Shown as CC in Figure 1-10, bits 18-19 can be set to a 0, 1, 2 or 3 depending on the result obtained in executing the last instruction. Most arithmetic and logical instructions, as well as some other instructions, set the condition code.

For example, the instruction **BRANCH ON CONDITION** can specify any selection of the condition-code values as a criterion for branching. Use the **BRANCH ON CONDITION** to test the contents of the CC of the current PSW that were set by the previous instruction.

The contents of the CC contains an address of another instruction (branch address) to be executed, depending on the comparison of the CC and the 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. The following are the types of codes found in the CC:

- ▶ Condition code (bits 18 and 19 of the PSW) set by some instructions and tell how those instructions ended.
- ▶ 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

Indicated by Prog Mask in Figure 1-10, bits 20-23 are associated with one of the following types of program exceptions:

- ▶ Fixed-point overflow (bit 20)
- ▶ Decimal overflow (bit 21)
- ▶ Hexadecimal floating point (HFP) exponent underflow (bit 22)
- ▶ HFP significance (bit 23)

During the execution of an arithmetic instruction, the PU may find some unusual (or error) condition, such as data overflow, loss of significance, or underflow. In such an instance, the relevant program mask bit is set to one (1) and a program interrupt occurs; refer to “Interrupts” on page 31 for more details.

When this program exception is encountered by z/OS, usually the current task is abnormally ended (ABEND). However, in certain situations, programmers do not want an ABEND for certain specific cases. In such a case, the programmer will instruct the program itself to handle the situation. So by using the unprivileged instruction **SET PROGRAM MASK (SPM)**, the interrupt can be bypassed by setting relevant program mask bits to off (0).

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 current PSW PU can be totally changed by two events:

- ▶ Loading a new PSW from CPC memory along an interruption.
- ▶ By executing the instruction LPSWE, which copies 128 bits from memory to the current PSW.

## Extended addressing mode

Indicated as EA in Figure 1-10, bit 31 controls the size of effective addresses and effective address generation along with bit 32, the basic addressing (BA) mode. 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 “Addressing mode” on page 30 for more information.

The combination of bits 31 and 32 identify the addressing mode (AMODE 24, 31 or 64) of the running program.

## Instruction Address

Bits 64 through 127, shown in Figure 1-10, point to the memory virtual address of the next instruction to be executed by this PU. When an instruction is fetched from CPC memory, its length is automatically added to this field. The PSW will then point to the next instruction address (see Figure 1-12). However, there are instructions, such as BRANCH, that may replace the contents of this PSW field, pointing to the branch target instruction. The address length contained in this PSW field depends on the program addressing mode attribute (AMODE) of the executing program. For compatibility reasons, old programs that address small addresses are still allowed to execute. When in 24- or 31-bit addressing mode, the leftmost bits of this field are filled with zeros. Then programs are allowed to run in 64-bit

addressing mode (AMODE 64), and they may also reside above the 2 GB address bar, under certain restrictions (field size on some used control blocks). In AMODE 64, a program can address data (operands) above the bar, and its instructions may reside above the line, as well. Refer to “A 64-bit Address Space” on page 51 for more details about line and bar concepts.

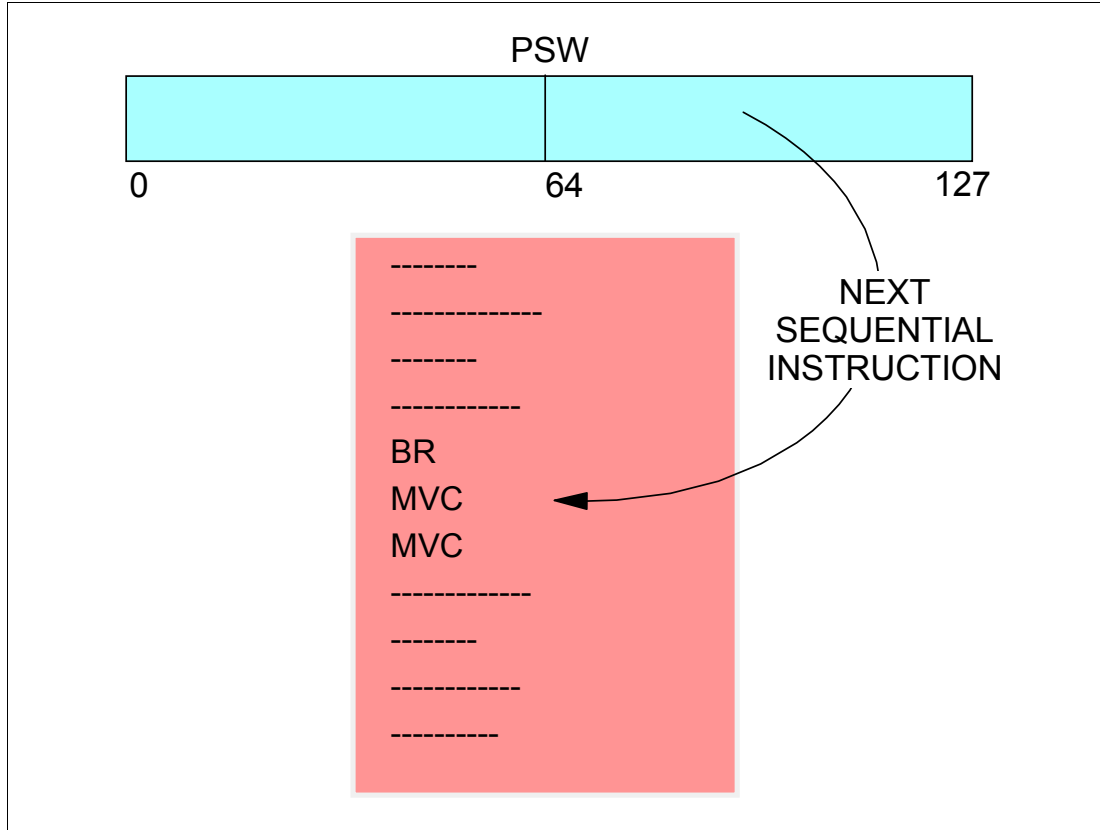


Figure 1-12 PSW next sequential instruction address field

The PSW bits 31 and 32 (extended and basic addressing modes) indicate to the PU which AMODE the running program is using.

The PU has an interrupt capability, which permits it to switch rapidly to another program in response to exceptional conditions and external stimuli. Interrupts can modify the address of the next instruction to be executed.

When an interrupt occurs, the PU places a copy of the current PSW in an assigned CPC memory location, called the old-PSW location, for the particular type of interrupt. Then, the PU fetches a new PSW copy from a second assigned CPC memory location. This new PSW (already prepared by z/OS) determines the next z/OS program to be executed (a z/OS component named First Level Interrupt Handler - FLIH). When it has finished processing the interrupt, the program handling the interrupt may reload the old PSW, making it the current PSW again, (through the LPSWE instruction) 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 CPC memory.

For more details about the interrupt concept, refer “Interrupts” on page 31.

## 1.11 Multiprocessing

Allowing many dispatchable units (tasks) at the same time in a system can cause a single PU to be heavily utilized. In the 1970s, IBM introduced a *tightly coupled multiprocessing complex*, allowing more than one PU to execute more than one tasks simultaneously (horizontal growth). All these PUs share the same CPC memory, which is controlled by a single z/OS copy in such CPC memory.

By the way, z14 implements Simultaneous Dual Thread, where two program tasks can be executed concurrently in the same PU (zIIP and SAP), each one in a Thread ID.

Summarizing, when you add more PUs to the CPC, you add the capability of processing program instructions simultaneously in these PUs. When all the PUs share CPC memory and a single z/OS image manages the processing, each task is assigned to a PU that is available to execute such task. If a PU has hardware failure, such tasks can be routed to another PU. This hardware and software organization is called a tightly coupled multiprocessor.

To implement tightly coupled systems, the following items are needed in the z/Architecture:

- ▶ Shared CPC memory, which allows hundreds of PUs to share the same logical partition CPC memory.
- ▶ PU-to-PU interconnection and signaling.
- ▶ Prefixing, implemented by the Prefix Register and related logic. This is beyond the scope of this volume.

## 1.12 z/Architecture Data Formats

CPC memory is a magnetic digital device used to store instructions and data manipulated by such instructions. CPC memory is made up of bits, 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. Then, z/Architecture allows different data formats depending on the nature of the information portrayed by the ones and zeros in memory or registers.

The following sections discuss data formats that are defined and used with the z/Architecture.

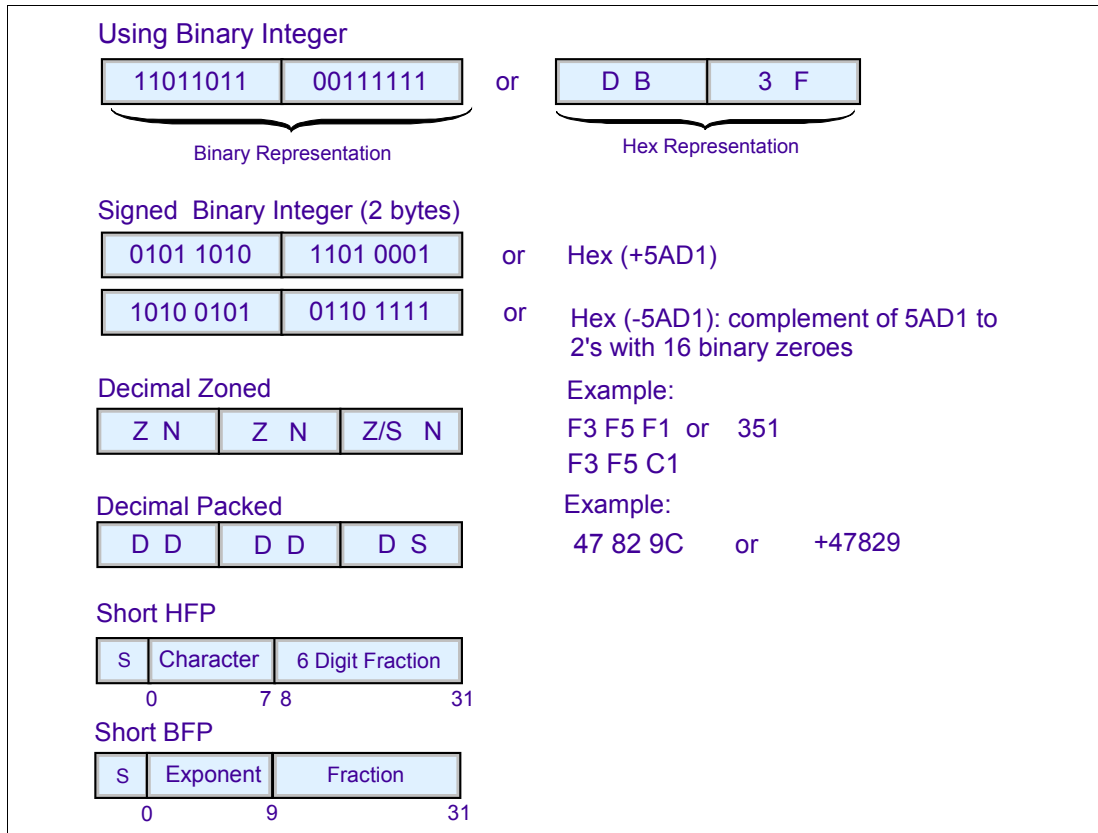


Figure 1-13 z/Architecture data formats

### 1.12.1 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 leftmost 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 halfword), four bytes (a fullword) or eight bytes (a doubleword). By the way, some floating-point instructions use extended operands made of sixteen bytes (a quadword).

### 1.12.2 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 (from 0 to 9), 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 (printers or screens). 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.

### 1.12.3 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 (DFP) 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 15. Recently, the z/Architecture introduced the Decimal Floating Point (DFP) format, very useful for commercial computing. At z14 there is also a hardware accelerator to execute instructions accessing DFP format.

### 1.12.4 EBCDIC data

Extended Binary Coded Decimal Interchange Code (EBCDIC) is an alphanumeric 8-bit (256 possibilities) code, developed by IBM. The code represents 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. GB

### 1.12.5 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.

### 1.12.6 Unicode

Unicode an alphanumeric double byte code with 64 KB possibilities, used by ideograms written language. It is fully supported in z/Architecture by a set of several instructions.

### 1.12.7 UTF-8

UTF-8 is an alphanumeric quad byte code with 4 GB possibilities, used by ideograms written language. It is fully supported in z/Architecture by a set of several instructions.

## 1.13 IPL of a z/OS logical partition

The initial program loading (IPL) hardware action provides a manual means for causing a program to be read from a designated device, load in memory and initiating its execution. At HMC one can choose which PU will perform the IPL, usually it is PU 0 by default.

An IPL is initiated manually by setting the load-unit-address controls to a four-digit device number to designate an input device. Before the z14, the IPL program ran in ESA/Architecture mode, changing to z/Architecture (via the SIGP instruction) when the Nucleus Initialization Program is activated. This is not possible any more, because the z14 only runs in z/Architecture.

## 1.14 CPC memory addressing

CPC memory is viewed as a long, horizontal string of bits, and formatted in multiples of 4 KB blocks, named frames. For most operations, access to memory 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. For the purpose of byte contents movement to PU caches, the CPC memory is divided in lines of 256 bytes each. Refer to Figure 1-14, which provides a diagram of CPC memory addressing.

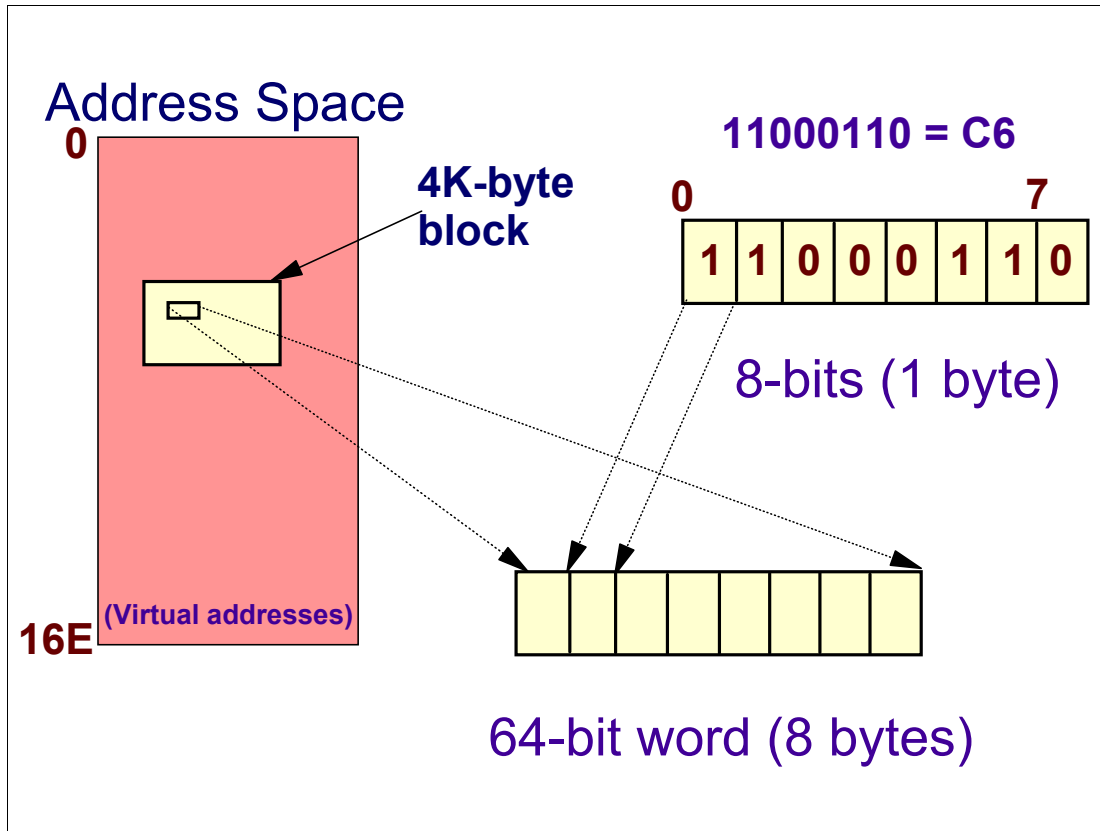


Figure 1-14 CPC memory addressing

### Bytes attribute

Each byte has two attributes:

- ▶ Location: Location in memory is identified by a unique non-negative integer, known simply as the byte address. Adjacent byte locations have consecutive addresses, starting with 0 on the left and proceeding in a left-to-right sequence. These addresses are located in fields that may have: 24, 31, or 64 bits.
- ▶ Contents: The given value 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. There are 256 different contents per byte.

In each byte, the bits are numbered in a left-to-right sequence. The leftmost bits are referred to as the “high-order” bits and the rightmost bits as the “low-order” bits. The bits in a byte are numbered 0 through 7, from left to right. Bit numbers are *not* memory addresses, however. Only bytes can be addressed. To operate on individual bits of a byte in memory, it is necessary to access the entire byte.

## 1.15 Addresses and address spaces

In this section, we discuss addresses and address spaces. Refer to Figure 1-15 on page 29 as you read this section.

In order to be executed, a program must reference CPC memory addresses. These addresses point to a set of bytes in CPC memory containing:

- ▶ An instruction, where its address is in the current PSW; refer to “Program status word (PSW)” on page 17, or
- ▶ 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 “Instruction set and its formats” on page 16), 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.

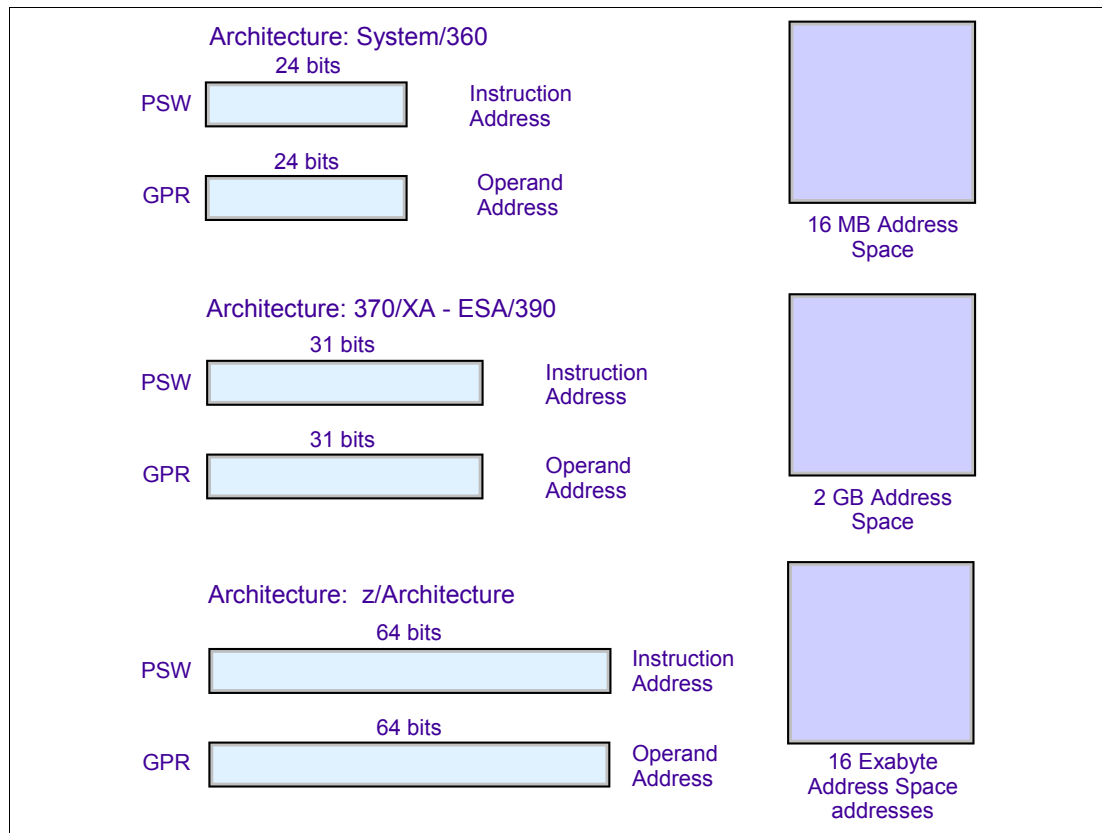


Figure 1-15 Addressing and address spaces

An address space (AS) is the set of addresses that this program may potentially reference during its PU execution. The number of possible addresses in an AS is controlled by the architecture of the PU which defines the size of the PU field containing such addresses. For example:

- ▶ System 360 architecture has 24-bit, which allows an AS of up to 16 MB addresses.
- ▶ System 370/XA architecture has 31-bit, which allows an AS of up to 2 GB of addresses.

In order to support the growth in business with large numbers of users and transactions, the z/Architecture has 64-bit, thus allowing each z/OS address space up to 16 EB of addresses.

In order to implement this, we need:

- ▶ 64-bit general registers and 64-bit control registers.

- ▶ To keep compatibility with old code, in addition to the 64-bit addressing mode, we have the 24-bit and 31-bit addressing modes which are carried forward. Any programs running with 24-bit or 31-bit addresses can still run in z/Architecture.
- ▶ The program status word (PSW) is expanded from 8 to 16 bytes to contain the larger next instruction address. The PSW also defines a newly assigned bit that specifies 64-bit addressing mode.
- ▶ For even more efficiency in virtual storage translation, we need up to three additional levels of dynamic-address-translation (DAT) tables, called region tables, for translating 64-bit virtual addresses. Refer to “Translating large virtual address” on page 47.

When virtual storage is not implemented, numerically the set of addresses in the AS is identical to the set of bytes addresses.

## 1.16 Addressing mode

An executable program (also known as a load module in z/OS) has an addressing mode (AMODE) and a residence mode (RMODE). These modes can be assigned to a load module at assembly (or compilation time), and at the binder processing time.

### 1.16.1 AMODE

The *addressing mode* determines where in virtual storage, the operands can reside. The memory operands for programs running in AMODE 64 can be anywhere in the 16 EB of addresses of the AS, while a program running in AMODE 24 can use only memory operands that reside in the first 16 MB addresses of the 16 EB AS.

One assigns an 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.
- 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 is restricted to only 24- and 31-bit addressing modes and does NOT include 64-bit addressing mode.

Figure 1-16 on page 31 is a graphical representation of PSW addressing mode bits.

### Running programs

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

The PU 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.

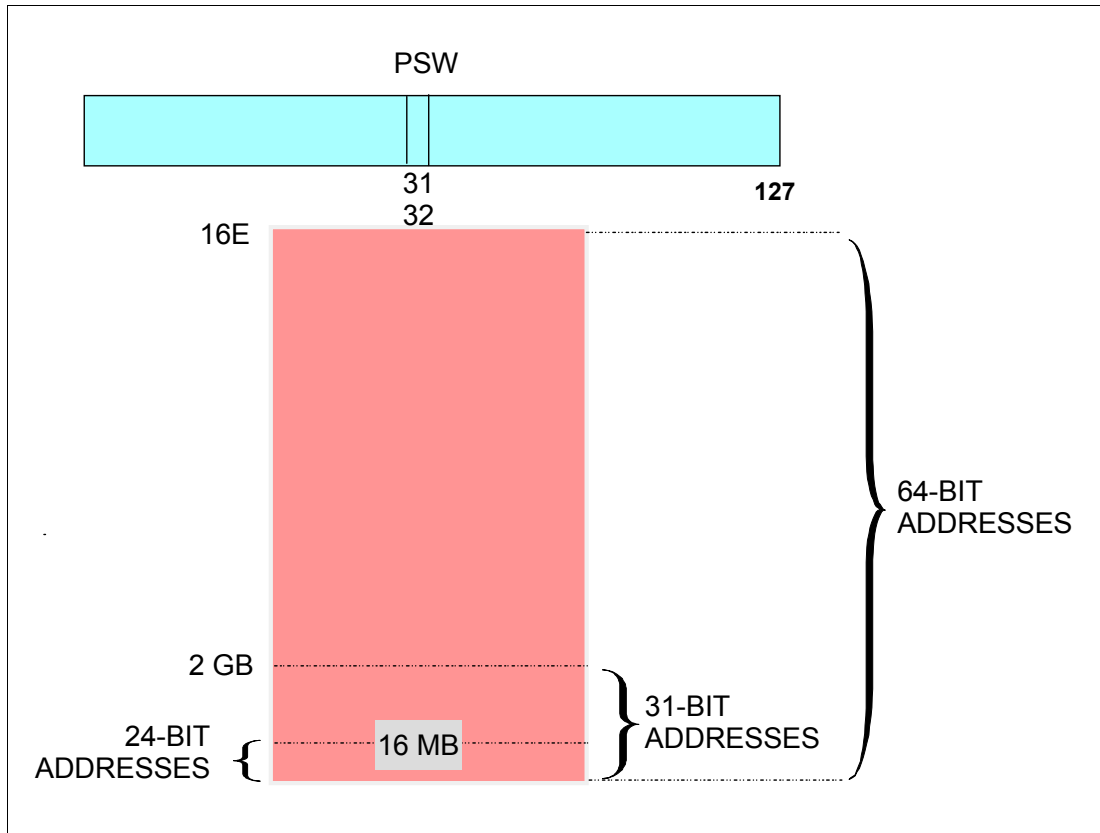


Figure 1-16 PSW addressing mode bits

### 1.16.2 RMODE

The *residence mode* indicates the location where the load module is stored. When brought into memory:

- 24** Indicates that 24-bit residence mode must be in effect, that is, store below 16 MB address. This address is also called, the line.
- ANY** Indicates that 24-bit or 31-bit residence mode must be in effect, that is, between the 16 MB and the 2 GB of addresses by preference.
- 64** Indicates that 64-bit residence mode must be in effect, that is, between above the 2G-bar. This residence mode is valid only at z/OS 2.3 under certain conditions.

## 1.17 Interrupts

An interrupt occurs in the PU when the PU detects an alteration in the sequence of instructions of a specific event or signal and are either planned or unplanned. The flow of these events is shown in Figure 1-17.

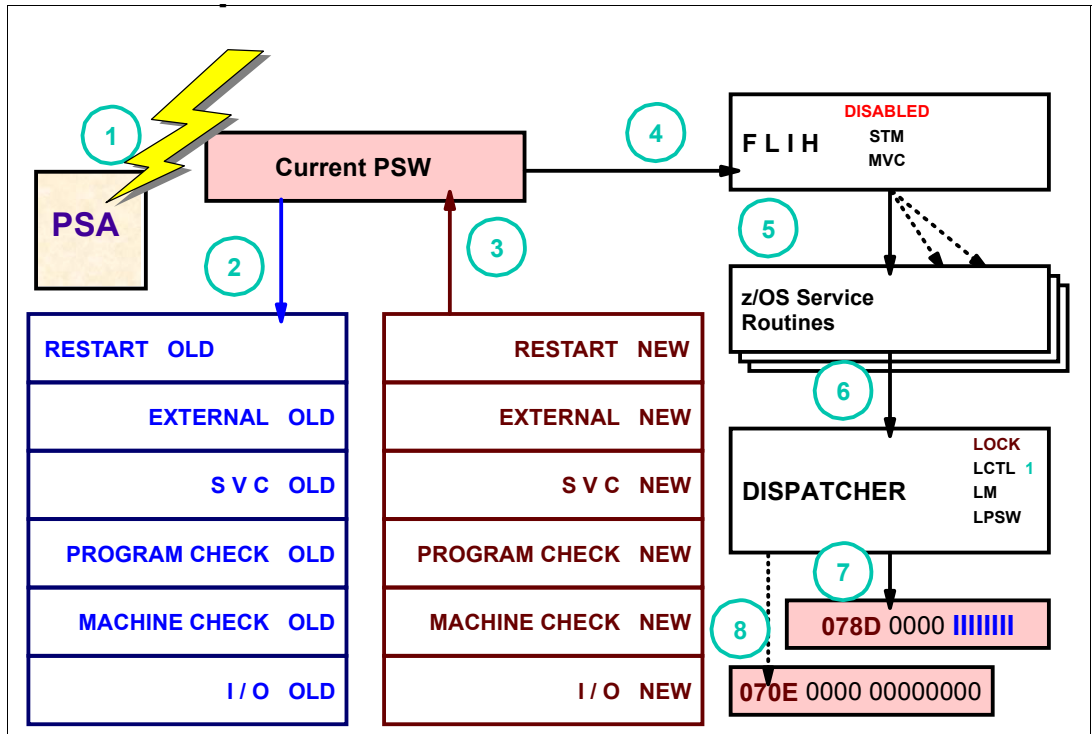


Figure 1-17 Interrupt process flow

In the Prefixed Storage Area (PSA) there are a pair of quadwords (made up of 16 bytes each) for each type of interrupt (1). When an interrupt occurs (2), an end of an I/O for example, the current PSW is stored in the I/O old PSW location (one of the quadwords for I/O interruption). Then the I/O new PSW (3) is loaded in the current PSW. That PSW points to (4) the corresponding First Level Interrupt Handler (FLIH). From the FLIH (5) control is passed to the service routines to handle the specific interrupt. After this is finished control is passed to the Dispatcher (6) that will look for work (7) and pass control (8) to the unit of work.

During interrupt processing, the PU hardware performs the three following hardware steps:

1. Stores (saves) the current PSW in a specific PSA CPC memory location named old PSW.
2. Stores information identifying the cause of the interrupt in specific PSA CPC memory location, named interrupt code.
3. Fetches, from a specific PSA CPC memory location named new PSW (already prepared by z/OS), an image of the PSW and loads it in the current PSW.

After this third step, the interrupt process is over. The PU returns to normal mode, that is, fetching the next instruction, where its virtual address is located at the current PSW (a new PSW copy).

In some IBM manuals, an interrupt is also called a PSW Swap.

**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).

## 1.17.1 Reasons for interrupts

When the PU 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 (add the length of the current instruction) 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 an alteration in the flow of control in a program. In an interrupt, however, one of the six interrupt types has occurred which needs to be brought to the attention of z/OS immediately, with a total different PU state. In the following section, we describe the flow of an interrupt including the repetition of the first three hardware steps.

## 1.17.2 Types of interrupts

In this section we discuss the six types of interrupts:

- ▶ Program check (or just program)
- ▶ Supervisor call
- ▶ Input / Output (I/O)
- ▶ External
- ▶ Machine check
- ▶ Restart (also called PSW restart at the HMC)

For more detailed information on these interrupt types, see:

[https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconc\\_interrupts.htm](https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconc_interrupts.htm)

### Program check interrupt

This interrupt is generated by the PU when something is wrong during the execution of an instruction. The description of such an error can be found in the Interruption Code stored in PSA memory during the PU hardware interruption process. For example:

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

Usually the z/OS (program FLIH/SLIH components) reaction to a program-check interrupt is to ABEND (abnormally terminate) the active current task associated with the program that 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 “PSW format” on page 18. 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 memory address does not correspond to a real address in CPC memory, then the task is not abended. Refer to “Dynamic address translation (DAT) (I)” on page 45, 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. However, with the wild branch hardware z/Architecture facility, the last address from which a successful branch instruction was executed is kept in memory location x'110'. This hardware facility is very useful in an assembler program debugging session.

### Supervisor call interruption flow

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

- ▶ X'0A' as the operation code, in the first byte
- ▶ SVC interrupt code (SVC number) in the second byte.

Figure 1-18 provides an example the flow of an SVC 0 interrupt.

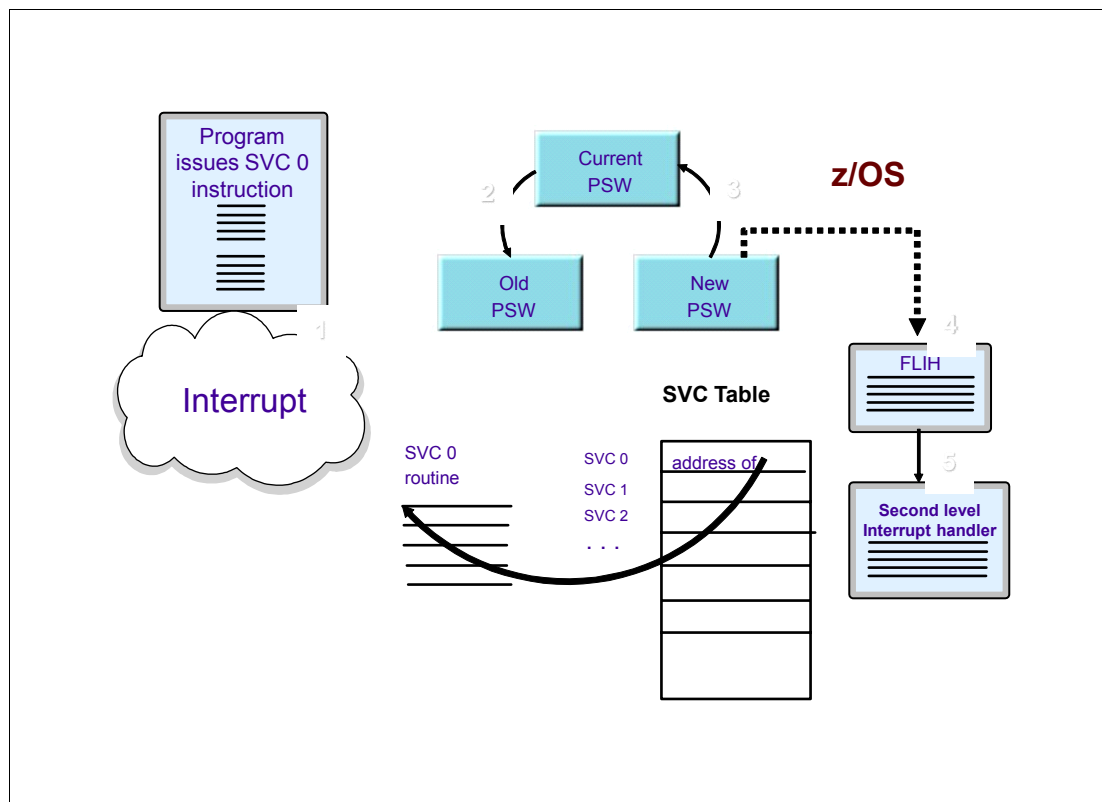


Figure 1-18 Example of an SVC 0 interrupt flow

When executed by the PU, the SVC instruction causes an SVC interrupt(1). The current PSW is stored in the PSA at the SVC old PSW address field(2), and the second byte of the instruction is stored in the SVC interruption code in PSA memory, and a new PSW (3) from the PSA new PSW address field is loaded in the current PSW.

The purpose of such an interrupt is a part of the z/Architecture, where an application program running in problem mode (bit 15 of the current PSW on) may pass control to z/OS asking for z/OS service. After the interrupt, the PU 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 supervisor call's (SVC) new PSW, which was prepared by z/OS and is in the supervisor state. The FLIH(4), already running in the supervisor state and saves the contents of the state (PSW and registers) in the Task Control Block (TCB), and other related control blocks. It then points to



the first instruction of a z/OS component named SVC first-level interrupt handler (FLIH) .In the majority of cases, the SVC FLIH branches to a second-level interrupt handler (SLIH) z/OS routine (5). SLIH gets the interruption code (SVC number), that is used as an index into a z/OS table named SVC Table.

There is an entry in such a table for every possible SVC number. The entry describes the attributes and the CPC memory address of a z/OS SVC routine that handles a required function associated with the SVC number. For example, SVC 00 means the z/OS Input Output Supervisor (IOS) component, 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 (contents of the second byte of the SVC instruction) and the z/OS component to be invoked. For example:

- ▶ 00 (EXCP) means that the application program is invoking the IOS component, asking for the execution by the channel of an I/O operation.
- ▶ 01 (WAIT) means that the active task wants to enter the wait state.
- ▶ 10 (GETMAIN) means that a program in the active task wants to have the right to access a certain number of virtual addresses.

After the request is processed by the z/OS SVC routine, the interrupted program can regain control again, by the z/OS Dispatcher component restoring its state (stored at TCB-related control blocks), that is, registers and the PSW through the LPSWE instruction.

## I/O interrupts

An input/output (I/O) interrupt occurs when the channel subsystem signals a change of status, such as I/O operation completion, an error occurrence, or an I/O device (such as a printer) has become available for work.

When an I/O operation is requested by a program in a task, the request is forwarded to the I/O supervisor (IOS) through an SVC 00 instruction (aa EXCP). After SVC interrupt processing, the SVC FLIH passes control to IOS. In z/Architecture, the I/O operation is not handled by the PU that executes z/OS code. There are less expensive and more specialized processors to execute the I/O operation, the I/O channels (usually FICON). When IOS issues the privileged START SUBCHANNEL (SSCH) instruction, the PU (CP or zIIP) delegates it to a SAP PU, that will find an I/O channel for the execution of the I/O operation. Then, the I/O operation is a dialog between this I/O channel and an I/O controller, in order to move data between CPC memory and the I/O device controlled by such a controller. After the execution of the SSCH instruction, IOS returns control to the program task issuer of the SVC 00. If needed, the access method (such as VSAM) still running in this task places itself in wait state (SVC 01), until the end of the I/O operation.

Now, how does IOS and the PU become aware that the I/O operation handled by the I/O channel is finished? This is handled through an I/O interrupt triggered by a SAP, and requested by the I/O channel at the end of the I/O operation.

Then, the I/O new PSW points to the IOS code in z/OS (I/O FLIH) and the interrupt code tells IOS which device has the completed I/O operation. Be aware that there are many I/O operations running in parallel. The final status of the I/O operation is kept in a hardware z/Architecture defined control block called the Interrupt Request Block (IRB).

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 program task is done, of course, by the z/OS dispatcher, gathering the status saved at TCB-related control blocks, and restoring it in the proper hardware registers, that is, executing a context switch.

The IBM z14, through PCIe HyperLink, introduces dramatic modifications in the flow of an I/O operation. These modifications are:

- ▶ FICON (zHPF) protocol is replaced by the standard PCI protocol, with a 5-times-faster data transfer rate (from 1.6 GB/sec to 8 GB/sec).
- ▶ For random 4-KB reads, with a hit in the DS8880 cache, the PU spins up to the end of the I/O operation that should take between 20 to 30 microseconds. As a consequence, such I/O operation does not cause an I/O interruption because it is synchronous.

### External interrupt

This type of interrupt has eight different causes, usually not connected with what the active program is doing. The most key causes are:

- ▶ 1004 Clock comparator - The contents of the TOD Clock became equal to the Clock Comparator; refer to “z/Architecture time facilities” on page 55.
- ▶ 1005 PU timer - The contents of the PU Timer became negative; refer to “z/Architecture time facilities” on page 55.
- ▶ 1200 Malfunction alert - Another PU in the multiprocessing complex is in checkstop state due to a hardware error. The address of the PU that generated the condition is stored at PSA locations 132-133.
- ▶ 1201 Emergency signal - Generated by signal PROCESSOR instruction when z/OS, running in a PU with a hardware malfunction, decided to stop (Wait Disable) that PU. The address of the PU sending the signal is provided with the interrupt code when the interrupt occurs. (**Note:** The PU 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 PU sending the signal is provided with interrupt code when the interrupt occurs. This function is used by z/OS to take a PU out of a wait state.
- ▶ 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 “z/Architecture time facilities” on page 55.

### 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 data integrity of system operation and to permit automatic recovery from some malfunctions.

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. Important to note that despite the error, The PU did not lose the capacity of executing instructions. The z/OS component in charge of the machine check interrupt is called Machine Check Handler (MCH).

### Restart interrupt

The restart interrupt provides a means for the operator (by using the restart function at HMC) or a program running on another PU (through a SIGNAL PROCESSOR instruction) to invoke the execution of a specified z/OS component program, that is the Recovery Termination Manager (RTM). It does an evaluation of the system status, reporting at the console: hangs, locks, and unusual states in certain key z/OS tasks. It gives the operator a chance to cancel the offending task. It is maybe the last chance to avoid an IPL.

The PU cannot be disabled for this interrupt.

### 1.17.3 Interrupt processing

The flow of interrupt processing is shown in Figure 1-19.

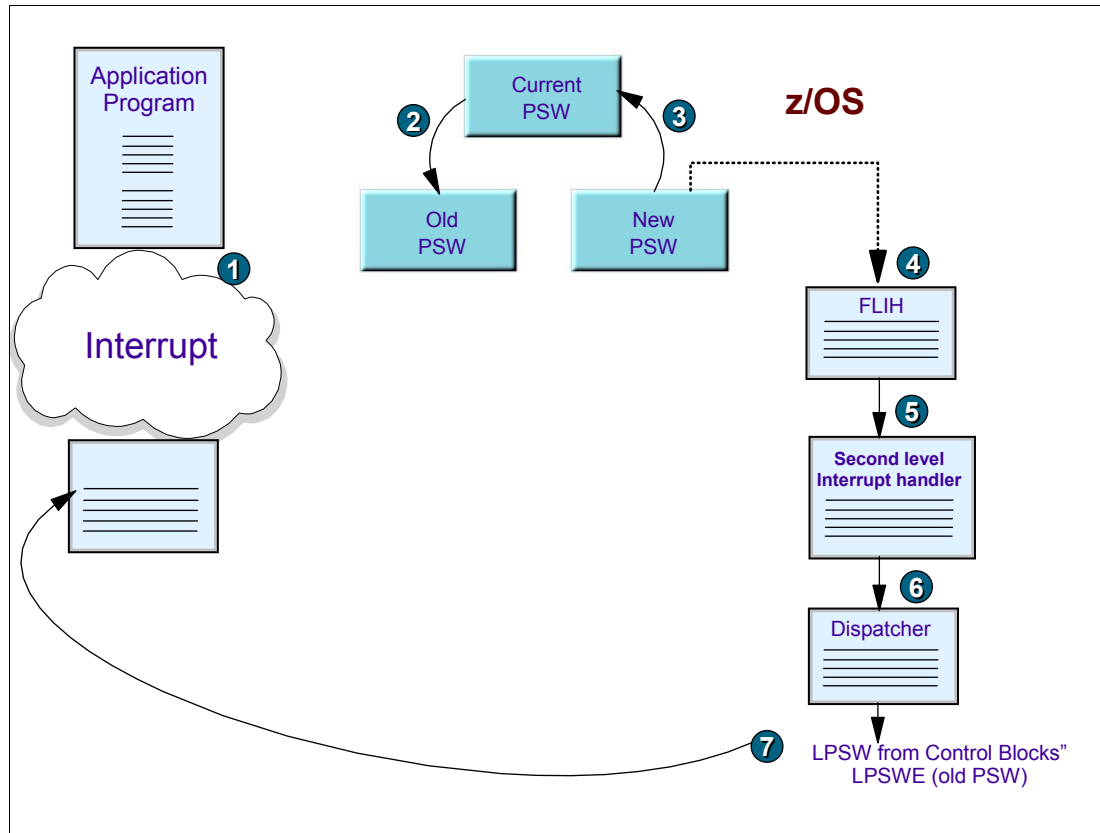


Figure 1-19 Software interrupt flow

The numbers in Figure 1-19 refer to the following:

**Step 1** One of the six types of interrupts is brought to the attention of the PU hardware. 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 for this type of interrupt (refer to “PSW format” on page 18), this interrupt is stacked in a systems assistance processor (SAP) hardware queue, and it will be taken care of as soon as this, or another PU, enters an enabled state.

**Steps 2 and 3** The PU is 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 into the current PSW. All the new PSWs point to one of the six z/OS component code first-level interrupt handlers (FLIH), whose duties are to save the running program status (PSW and registers) and then select an appropriate z/OS service routine, also called second-level interrupt handler (SLIH) to pass control of the PU. FLIH saves the interrupted task state in specific control blocks, all related to the Task Control Block (TCB).

All of these actions are also known as Context Switch.

**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?

**Answer:** A branch cannot, and does not, change the status bits of the current PSW, such as the bit 15, the problem/supervisor bit.

- Step 4 Control is then passed to the first-level interrupt handler (FLIH), a z/OS component. z/OS will use privileged instructions to respond to the event; the new PSW (prepared by z/OS itself during nucleus initialization program (NIP) processing) has bit 15 turned off (refer to “Problem or supervisor state” on page 21). z/OS needs to access some CPC memory 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 memory locations. Refer to “Storage protection” on page 39, for more information on PSW key field.
- Step 5 After saving the status, 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.
- Steps 6 and 7 The Dispatcher (a z/OS component) dispatches (delivers the PU to a program of the selected task) the interrupted program if there is an available PU that is ready for new work. It might dispatch only the interrupted one or any other, depending on the dispatching priority queue. Recall that FLIH saved the status (PSW and registers) on control blocks, such as TCBs. The z/OS Dispatcher does this by using the old PSW copy, which has been saved, and which contains PU status information necessary for resumption of the previous interrupted program task. 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 at the exact point.

## 1.18 Prefix Storage Area (PSA)

PSA is a PU mapping 8-KB CPC memory starting at the CPC memory location zero.

The function of the PSA is to be a memory communication area between the operating system (z/OS) and the PU hardware. This communication is made in the two directions. For example, the field named old PSW is information from the hardware to z/OS and the PSA field named new PSW is information from z/OS to the hardware.

Figure 1-20 depicts the layout of the PSA in z/Architecture.

END. Length . Function			END. Length Function		
0	8	Restart <b>NEW PSW; IPL PSW</b>	149	1	<b>Monitor class</b>
8	8	Restart <b>OLD PSW; IPL CCW1</b>	150	6	<b>PER (1 or 2) Code + PER Address</b>
16	8	<b>CYT address; IPL CCW2</b>	156	4	<b>Monitor Code</b>
24	8	External <b>OLD PSW</b>	160	2	<b>Exception Access + PER Access</b>
32	8	Supervisor Call <b>OLD PSW</b>	184	4	<b>SID (0001+Subchannel #) =&gt; 3.1</b>
40	8	Program Check <b>OLD PSW</b>	188	4	<b>I/O Interr.Param.subchannel (@ UCB)</b>
48	8	Machine Check <b>OLD PSW</b>	216	8	St.Status / Mach.Check <b>CPU Timer SA</b>
56	8	Input / Output <b>OLD PSW</b>	224	8	St.Status / Mach.Check <b>Clock Comp.SA</b>
88	8	External <b>NEW PSW</b>	232	8	<b>Machine Check</b> Interruption Code
96	8	Supervisor Call <b>NEW PSW</b>	244	4	External <b>Damage Code</b>
104	8	Program Check <b>NEW PSW</b>	248	4	<b>Failing</b> Storage Address
112	8	Machine Check <b>NEW PSW</b>	256	16	St.Status <b>PSW SA</b> ; Fixed <b>Logout</b> area
120	8	Input / Output <b>NEW PSW</b>	272	16	Reserved
128	4	<b>External</b> Interr.Parameter	288	64	St.Status / Mach.Check <b>Access reg.SA</b>
132	4	<b>CPU Address + External Code</b>	352	32	St.Status / Mach.Check <b>Flt.Pt. reg.SA</b>
136	4	<b>SVC Interruption: ILC + Code</b>	384	64	St.Status / Mach.Check <b>General reg.SA</b>
140	4	<b>Program Interruption: ILC + Code</b>	448	64	St.Status / Mach.Check <b>Control reg.AS</b>
144	4	<b>Translation</b> Exception ID			.....

Figure 1-20 PSA layout

## 1.19 Storage protection

With multiprocessing, hundreds of tasks can run programs, accessing physically any piece of CPC storage (or memory). z/OS preserves the integrity of each user's work through the use of storage protection, also known as memory protection. Its main purpose is to prevent a process from accessing storage (or memory) that has not been allocated to it.

This is defined by the z/Architecture to protect CPC memory. This facility is complemented by the virtual storage mode (PSW bit 5 on).

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

All real addresses manipulated by PUs or I/O channels must go through storage protection verification before being used as an argument to access the contents of CPC memory. The input of memory protection is a CPC memory address, the PSW key field, and the output is either an 'OK' or a program interrupt with interruption code X'0004'.

### 1.19.1 Storage key

For each 4 KB block of CPC memory (frame), there is a 7-bit control field, called a storage *key*. This key is used as follows:

- ▶ 4 access control bits (aka protection key), shown as KKKK in Figure 1-21 on page 40. These four bits may describe 16 different protection keys (from 0 to F). The protection key is associated with the page being associated with such frame. Each page acquires a protection key at the Getmain moment depending on its subpool number.
- ▶ Fetch bit, shown as F in Figure 1-21. 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, shown as R in Figure 1-21. Bit 5 is associated with dynamic address translation (DAT). It is normally set to one whenever a location in the related 4 KB memory block is referred to for either storing or fetching of information.
- ▶ Change bit, shown as C in Figure 1-21. 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 memory.

### 1.19.2 PSW key field

- ▶ The PSW key field (bits 8 to 11 and not shown in the figure) is set by z/OS through the privileged SET PSW KEY FROM ADDRESS instruction. This key is compared to the access control bits (kkkk) for the frame being referenced for memory access.

The reference and change bits do not participate in the memory protection algorithm. They are used for virtual memory implementation; refer to “Dynamic Address Translation (DAT) (II)” on page 46.

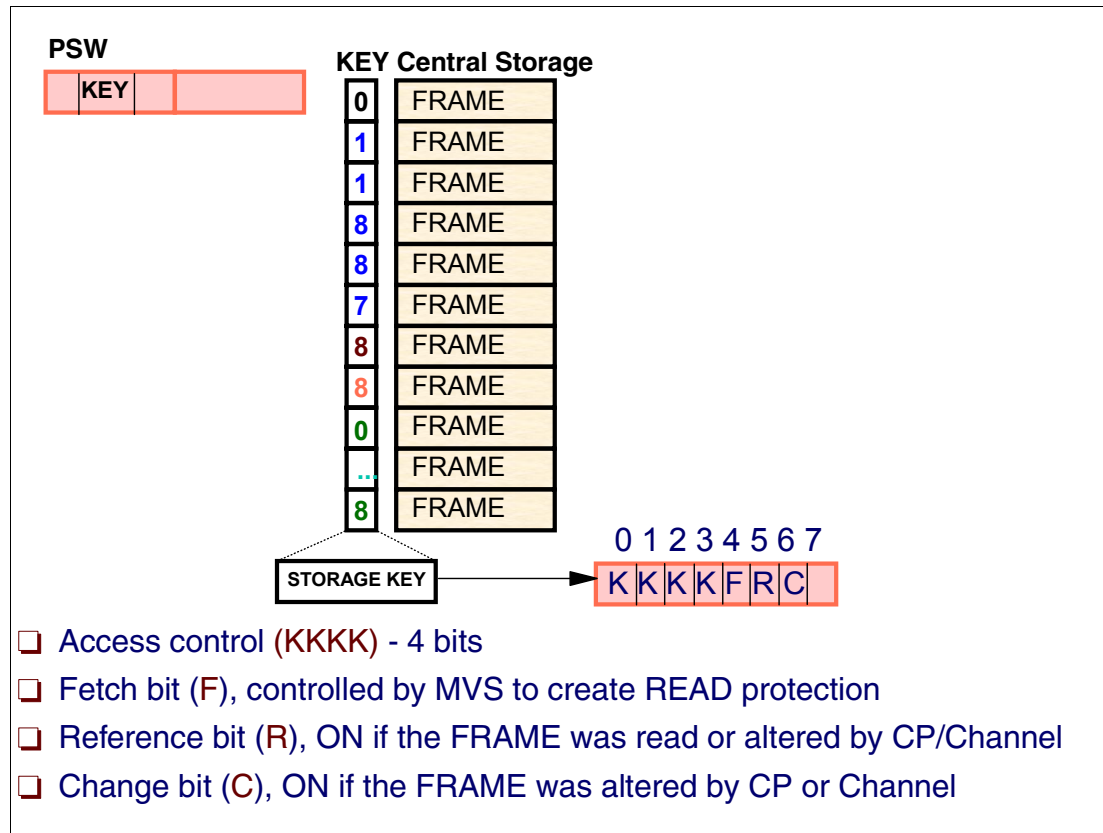


Figure 1-21 Storage Protection elements

### 1.19.3 Storage protection logic

z/OS may alter storage key bits by issuing an instruction, and will inspect them using other instructions. Storage protection is invoked when CPC memory is going to be accessed through a real address (for fetch or store) by a program running in a PU.

The reference bit is inspected and switched off after inspection. The PU storage hardware switches on the reference bit when the frame is accessed by a PU or any channel, and also switches on the change bit when the frame contents are changed by those components (see Figure 1-22). The reference and change bits do not participate in storage protection. They are used for virtual storage implementation.

The following conclusions can be reached from storage protection logic:

- ▶ If a running program has a PSW key equal to 0000, it may access any frame in memory.
- ▶ If the fetch bit is off in a frame, any program with any PSW key 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.

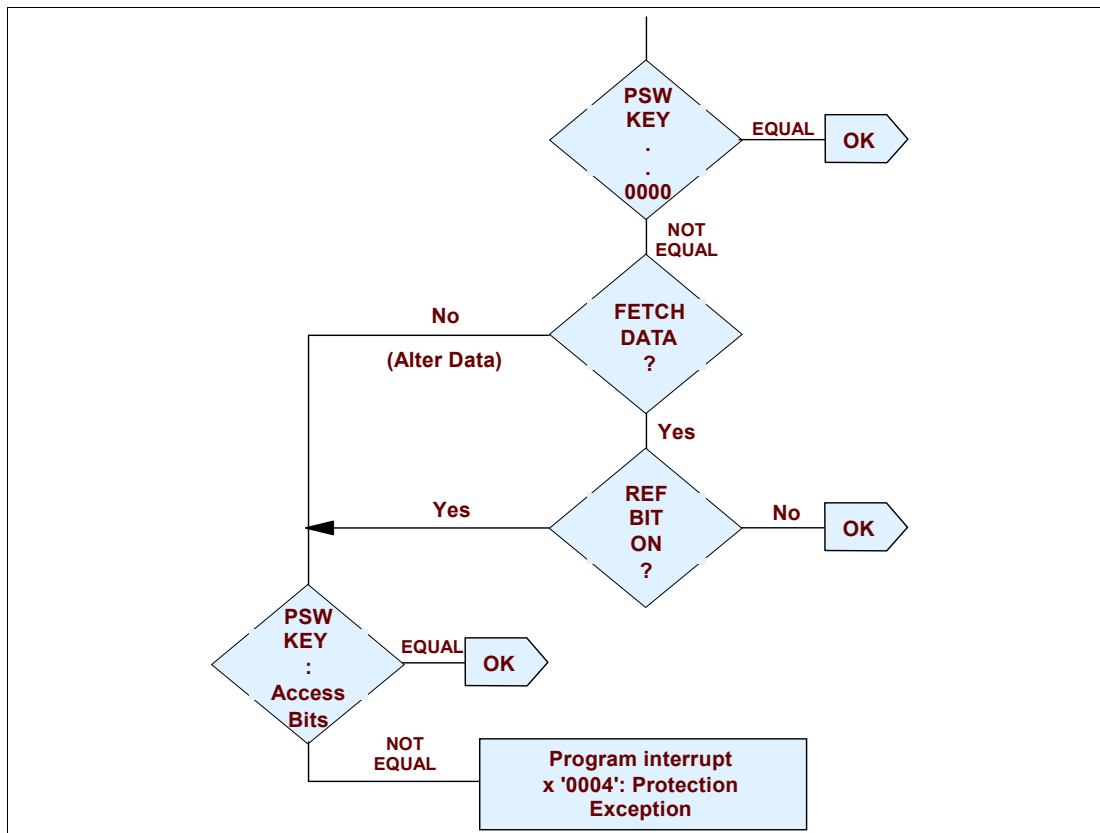


Figure 1-22 Storage protection logic

z/OS exploits storage protection by managing frame memory 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 some z/OS subsystems run with PSW key one. Application code has PSW key eight.

It is important to note that this protects the address space of the common areas that are accessed by programs running in different address spaces. The address space private areas are naturally protected by different segment (or region) tables. Refer to “Dynamic address translation (DAT) (I)” on page 45.

## 1.20 Virtual Storage initial concepts

Virtual storage is a z/Architecture illusion, where each program believes that there are more bytes in CPC memory than there really are. For example, z/OS running in a logical partition allows for CPC memory up to 4 TB (in real addressing), however, a program may reference up to 16 EB virtual addresses located in an AS.

This illusion is created by the z14 hardware PU component named Dynamic Address Translation (DAT) along with other z/OS components such as:

- ▶ The virtual storage manager (VSM),
- ▶ The real storage manager (RSM) and,
- ▶ The auxiliary storage manager (ASM)

Before virtual storage, the number of programs that could run in CPC memory was restricted to how much CPC memory there was. For example, if you had 100 KB of CPC memory, only two programs of 50 KB each would fit in memory. And in this 50 KB program, only 10 KB contained the active portion of the program.

This was not efficient because, by the principle of locality, only less than 20% of a program (load module) is frequently executed (kernel). So, it is a waste of CPC memory keeping rarely executed code in memory. On top of that, the transaction multiprogramming level (number of current tasks) is very low due to the limitation in the number of concurrent programs. Another key consequence would be that a very low average CPU utilization.

With virtual storage, the code that is not currently active is stored in page data sets, making room in CPC memory for the active portions of each program to be loaded.

The management of address spaces by z/OS is in various sizes. Main memory is partitioned into equal fixed size chunks that are relatively small and each process is also divided into small fixed size chunks of the same size. Then, the chunks of a process, known as pages, are assigned to available chunks of memory, known as frames or page frames. For example, page address spaces are divided into 4 KB units of virtual storage called pages. A page fault occurs when the page containing the PU referenced address is not associated with a memory frame.

Segment address spaces are divided into 1-megabyte units called segments. A segment is a block of sequential virtual addresses spanning megabytes, beginning at a 1 MB boundary. A 2 GB address space, for example, consists of 2048 segments.

A region is a block of sequential virtual addresses spanning 2-8 GB, beginning at a 2 GB boundary. A 2 TB address space, for example, consists of 2048 regions.

*Virtual addresses* are the addresses contained in the AS, identifying a specific address in the AS. When a virtual address is used by the PU for an access to CPC memory, it is translated by means of DAT to a real address. By the way, DAT is a circuit within each PU. A real address identifies a location of a byte at CPC memory.

Refer to Figure 1-23 on page 43, where you may see 8 pages from an AS distributed among frames at CPC memory and slots at page data sets.



It is possible, on top of the 4 KB page, to have 1 MB and 2 GB pages depending on the exploiting software accessing the virtual storage. The large pages improve DAT performance.

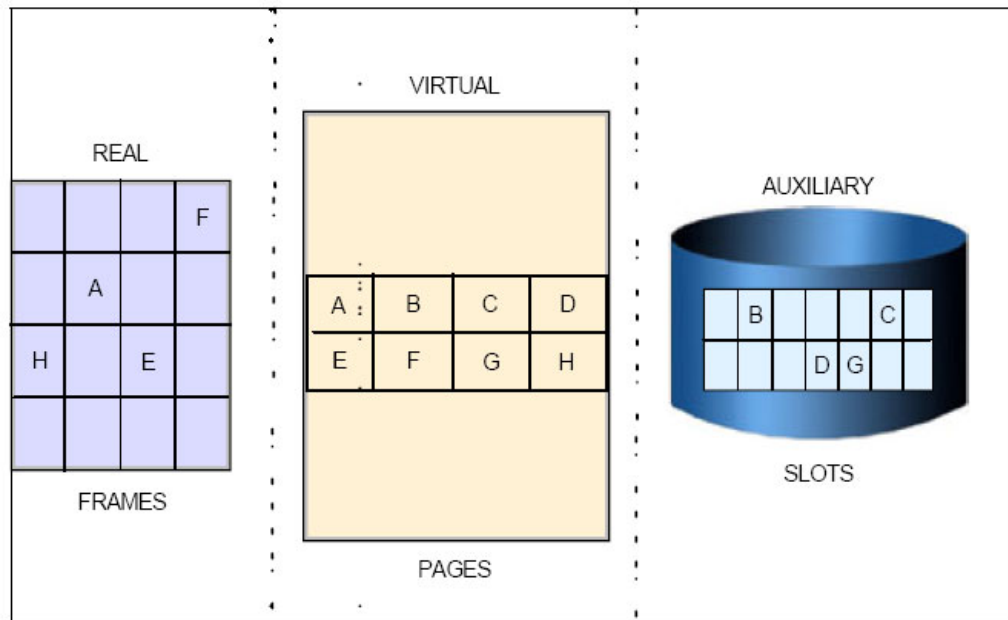


Figure 1-23 Pages, frames and slots

### Addressing bytes in CPC memory

CPC memory provides the PU with directly addressable fast access memory bytes of data and programs. They must be loaded into CPC memory (coming from I/O input devices) before they can be processed. The CPC memory is available in multiples of 4 KB blocks named frames.

PU's and I/O channels in a z14 machine access CPC memory byte contents through Memory Controller Units (MCU) located at the PU chip that connects to CPC memory boxes (DIMM). PU's and I/O channels send byte addresses to MCUs and these return byte contents. This behavior is called non-associative.

## 1.21 Segmenting a virtual address

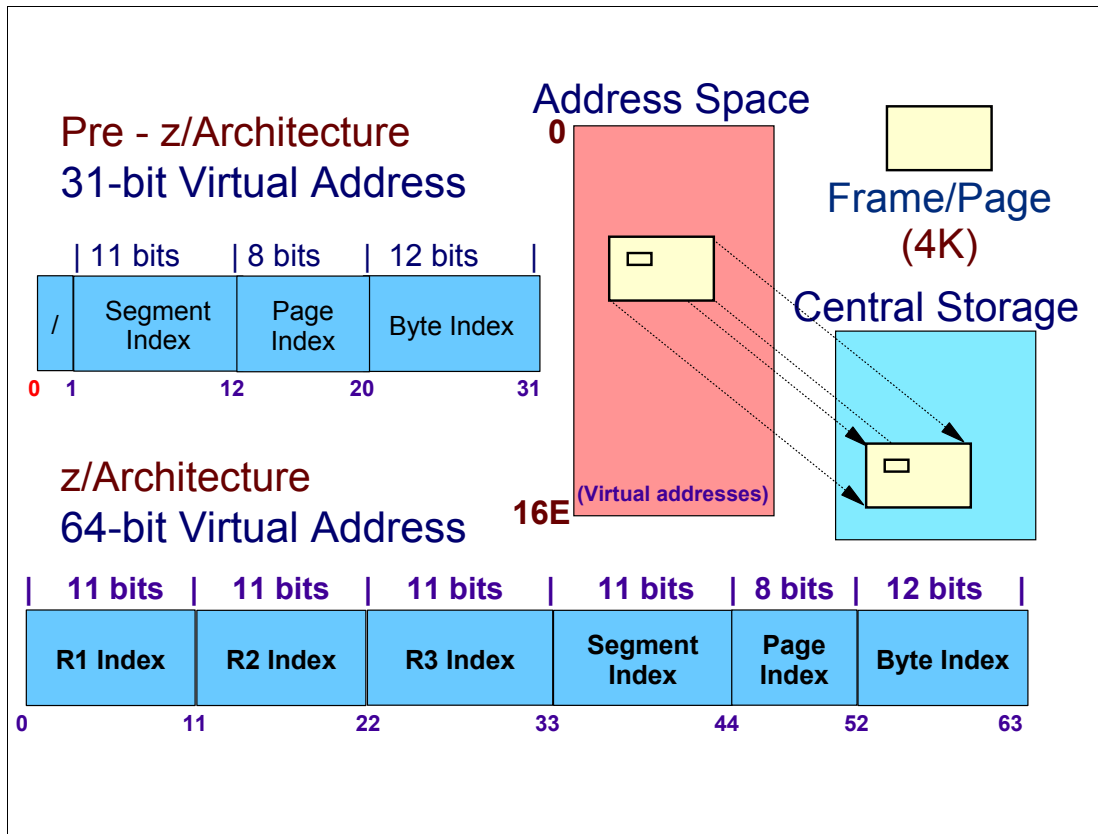


Figure 1-24 Segmenting a virtual address

### Segmenting a virtual address

In order to translate a virtual address into a real address, DAT must know for the 64-bit virtual address:

- ▶ The number of the first region where this virtual address is located (R1 index- 11 bits).
- ▶ The number of the second region where this virtual address is located (R2 index- 11 bits).
- ▶ The number of the third region where this virtual address is located (R3 index- 11 bits).
- ▶ The number of the segment where this virtual address is located (SI index- 11 bits).
- ▶ The number of the page where this virtual address is located (PI index- 18 bits).
- ▶ The displacement within the page (BI - 12 bits).

## 1.22 Dynamic address translation (DAT) (I)

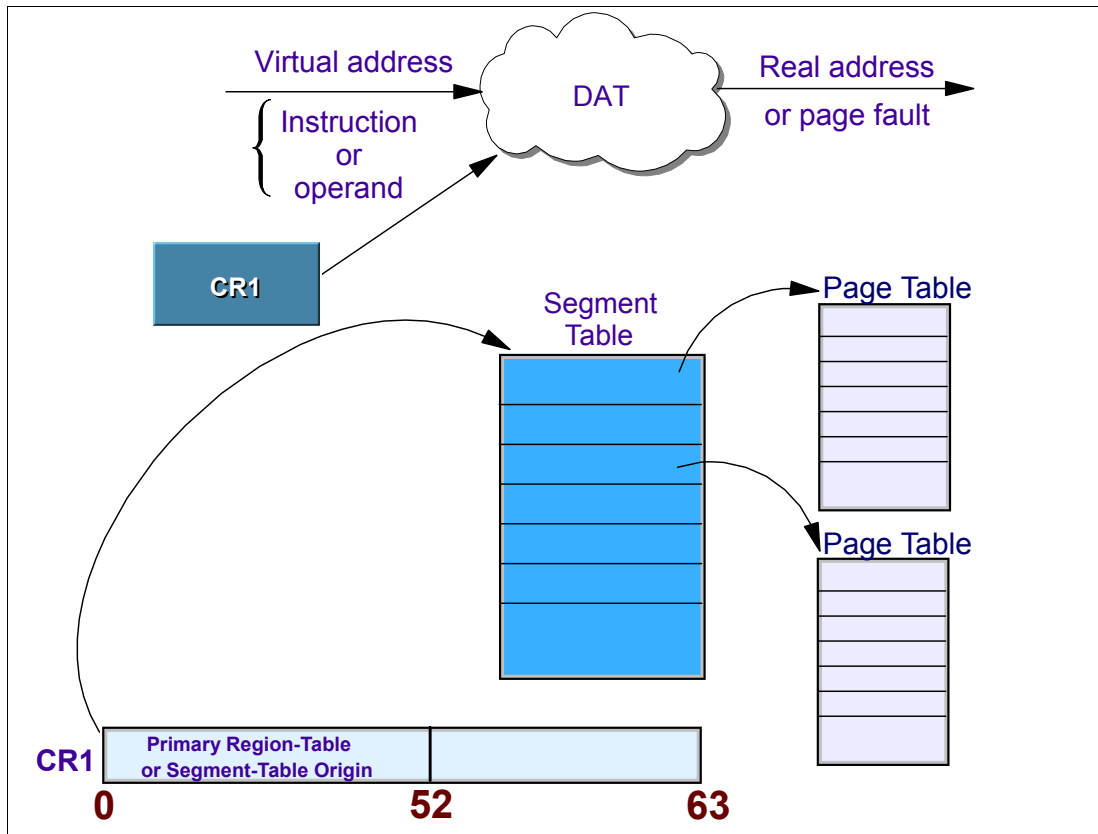


Figure 1-25 Translating a virtual address up to 2G

### Control registers and translating tables

DAT receives a virtual address of an instruction or an operand and translates such address. The outcome might be the real address in the page-corresponding frame at CPC memory or a program-interrupt named page fault, with x'11' as an interruption code.

In order to accomplish such task, DAT access translation tables are maintained by the z/OS Real Storage Manager (RSM) component. There is a set of tables for each z/OS address space (AS). The first-layer table depends (region table or segment table) on the high-water-mark requests virtual storage using the GETMAIN macro.

Assuming that the virtual address to be translated is up to 2 GB, the high-level table corresponding to the active AS is a segment table. This table is pointed to by a special register named control register 1 (CR1). RSM is in charge of maintaining such CR1. There is one segment table per each current AS.

The predecessor of today's z/OS was called Multiple Virtual Storage (MVS), meaning multiple address spaces. Whenever the ATTACH macro creates a task, it is decided if the programs of this task will share virtual addresses with already existing programs of other tasks. If the answer is yes, the newborn programs will run in an already existing AS. If not, a new AS is also created during task creation, with CR1 pointing to a new segment table when it becomes active.

## 1.23 Dynamic Address Translation (DAT) (II)

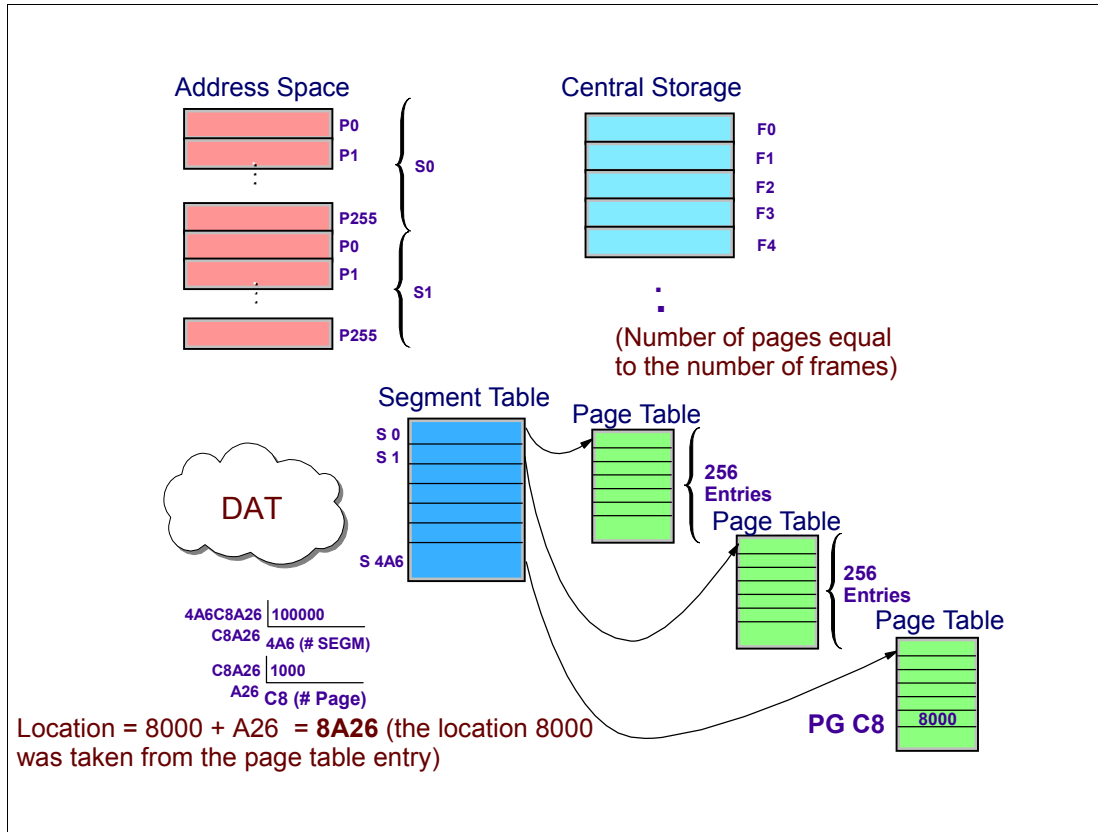


Figure 1-26 Translating a virtual address up to 2G

### Translating a virtual address up to 2G

DAT is the hardware (within the PU) in charge of translating a virtual address during a CPC memory reference into the corresponding real address. In order to accomplish this, DAT accesses indexed tables called region, segment, and page tables. These tables are maintained by a z/OS component called Real Storage Management (RSM).

#### z/OS translation tables

When the virtual address to be translated is up to 2 GB, then there are two translating tables only: a segment table, and a page table. Each entry in a segment table points to a page table. Each entry in the page table points to the location of the memory frame associated with that page.

DAT performs the following tasks for a virtual address space that is less than 2 GB:

- ▶ Receives a virtual address from the PU. It does not matter whether it refers to an operand (data) or to an instruction.
- ▶ Segments the virtual address as described in “Segmenting a virtual address” on page 44. It does that by dividing the virtual address by 1 MB. The quotient is the number of the segments (S), and the remainder, if any, is the displacement within the segment (D1).
- ▶ Finds the corresponding entry (S) in the segment table to obtain the pointer of the corresponding page table.

- ▶ Divides the D1 segment by 4 KB. The quotient is the number of the page (P) in that segment, and the remainder, if any, 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 PU to allow access to the CPC memory contents.

Figure 1-26 on page 46 illustrates the process of translating the address x '4A6C8A26'.

## 1.24 Translating large virtual address

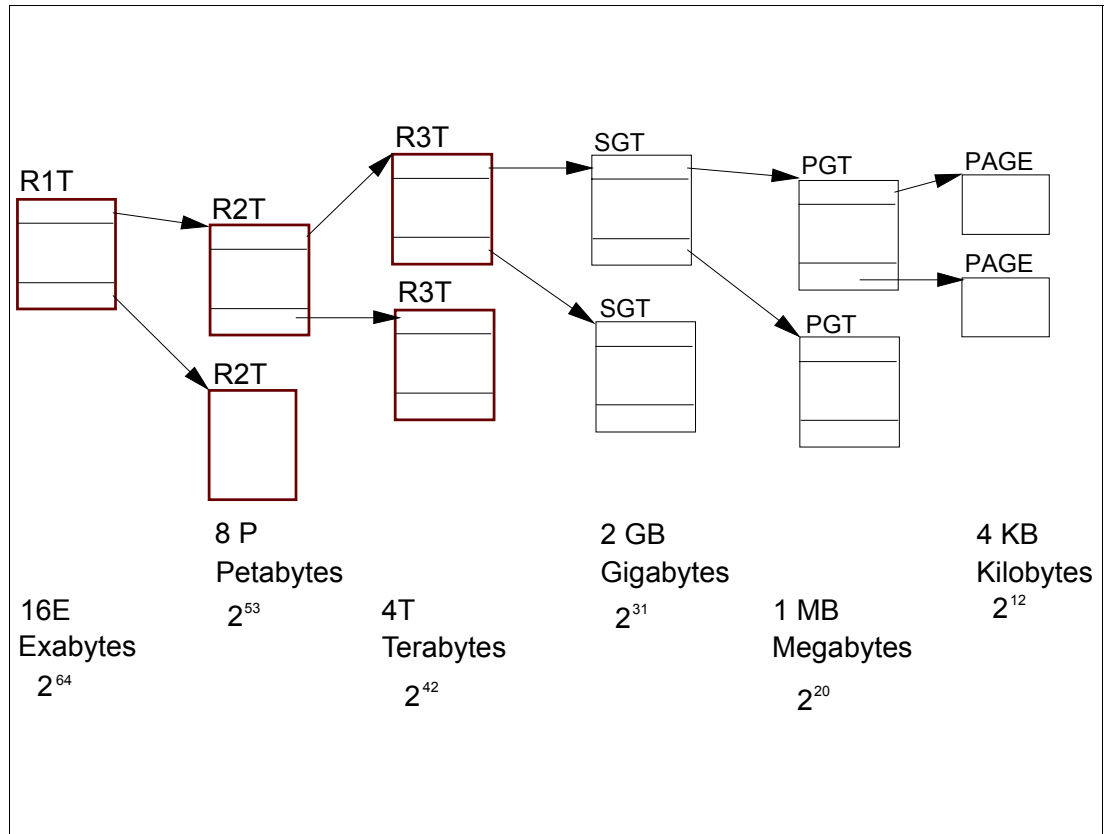


Figure 1-27 Region, segment and page tables pointing

### Region tables for 64-bit

There are up to five levels of translating tables for a virtual address larger than 2G:

- First region table
- Second region table
- Third region table
- Segment table (also used for virtual addresses less than 2 GB of addresses)
- Page table (also used for virtual addresses less than 2 GB of addresses).

Each entry in a region table may point to another region table or to a segment table (depending on the region table layer), and the segment table always points to a page table.

Then, in a 16-EB AS with 64-bit virtual memory 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 size, and there are 2048 entries per table.

An AS in z/OS is born with 2 GB of addresses only. When the first Getmain above 2 GB (bar) memory is requested, RSM creates the R3T. (Refer to “A 64-bit Address Space” on page 51 for more details about the bar concept.) The R3T table has 2048 segment table pointers, and provides addressability up to 4 TB virtual addresses. When virtual memory Getmain address received is greater than 4 TB, an R2T is created. An R2T has 2048 R3T table pointers and provides addressability to 8 PB virtual addresses. An R1T is created when virtual memory getmained greater than 8 PB is allocated. The R1T has 2048 R2T table pointers, and provides addressability to 16 EB virtual addresses. At Figure 1-27 on page 47, you may see the table layers and the respective pointers.

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

RSM only creates the additional layers of region tables, when necessary to back virtual storage that is mapped by future Getmains. Then, they are not built until needed. So, just as an example, if an application requests 60 PB of virtual memory, the necessary R2T, R3T, segment table, and page tables are created because there is a need to back a referenced page.

Summarizing, up to five translation tables may be needed by DAT to do translation, but the translation only starts from the table layer that provides translation for the highest getmained virtual address in the AS.

## 1.25 Page faults and page data sets

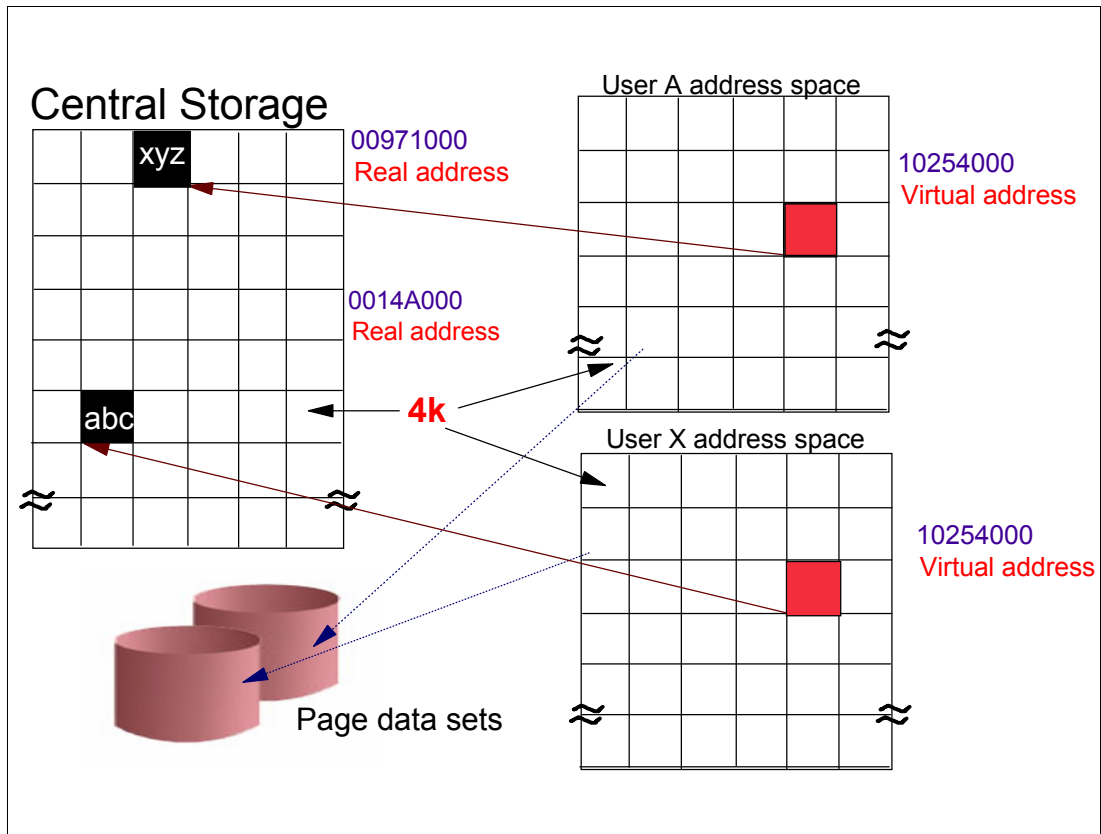


Figure 1-28 Page faults and page data sets

### Page faults

Following the DAT processing of a virtual address, either a real address is returned 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 CPC memory. DAT reacts to this by generating a program interrupt code X'11" indicating a page fault. This page must be read in from external memory called a page data set located in a 3390 volume or in z14 Virtual Flash Memory (VFM).

### Page data sets

Page (or paging) data sets are 3390 data sets managed by the z/OS component Auxiliary Storage Management (ASM). They contain the paged-out portions of all virtual memory AS (and data spaces). Before the first IPL, an installation must allocate sufficient space on paging data sets to back up the following virtual memory areas:

- ▶ Space for the pageable portions of the common area (PLPA and Common page data sets)
- ▶ Space for all address spaces private areas (Local page data sets)
- ▶ VIO data sets that are backed by auxiliary memory (Local page data sets). VIO data sets are temporary and not large data sets mapped at virtual storage only.

### Address space virtual memory

Figure 1-28 on page 49 shows two address spaces with a same virtual memory address of 10254000 which contains either data or executable code belonging to each of these AS.

Although these virtual addresses are numerically identical, they belong to these AS private areas. It means that the containing pages are mapped at different CPC memory frames.

For each AS to reference the 4 KB page beginning at that address, that 4 KB page must be mapped with a 4 KB frame at CPC memory. When referenced, the page could either be in CPC memory already, or a page fault occurs and it must be read from the page data set where it resides, through an I/O operation.

## DAT and z/OS

To determine how to access the page in an AS, the functions are divided between hardware (DAT) and z/OS (RSM and ASM components), as follows:

- ▶ DAT is in charge of translating the *virtual address*, producing a *real address in CPC memory*, 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 PU. When off, it means that the virtual address is equal to the real address and DAT is not invoked. In z/OS, with the exception of the execution of a small routine at the nucleus, the bit 5 is always ON.

- ▶ z/OS (RSM and ASM) is in charge of the following tasks:
  - Maintaining the tables (regions, segment and page tables).
  - Deciding which page contents are kept in CPC memory frames. It does that by executing page stealing, when the number of available frames falls below a certain minimum threshold. The pages to be stolen are the least referenced, the algorithm to keep such pages is called Least Recently Used (LRU). Pages in a CPC memory frame have an associated count called the *unreferenced-interval count* (UIC) that is calculated by RSM. This count measures how many seconds a page is unreferenced. The pages with the highest UIC counts are the ones to be stolen.
  - Allocating I/O DASD page data sets (at 3390 volumes or VFM) to keep unreferenced pages, when there is a CPC memory shortage. These data sets are formatted in 4 KB slots and managed by ASM.
  - Supporting page faults, that is bringing to a CPC memory frame the copy of the page-faulted page from the page data sets. This I/O operation is called page-in.

## More about maintaining address space tables

z/OS creates a segment table for each new AS, that at its beginning has only 2 GB virtual addresses. The segment table entries point to page tables for only the ones representing pages with valid data (returned from a Getmain) in that segment. Figure 1-27 on page 47 shows this pointing structure. If a segment has no valid data in it, there is no need to build and maintain its page table. There is a bit in segment table entry (the invalid bit) that indicates this condition to DAT.

- ▶ DAT knows which is the current AS because z/OS (RSM) loads, in control register 1, the real address of its segment table. Then, each AS has a:
  - Private area, where same numeric virtual addresses in different address spaces maps to distinct CPC memory frames. As shown in, Figure 1-28 on page 49, the same virtual address 10254000 is referred to in two different address spaces that maps to distinct real addresses. Each virtual address is translated by its own segment table, whether up to 2 GB.
  - Common area, where same numeric virtual addresses in different address spaces maps to the same CPC memory frame. This common area is used as a communication between program running in different address spaces. 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 AS, the segment table is not pointed by the control register 1. In this case such registers points to the appropriate region table: third region table, second region table and first region table; refer to “Translating large virtual address” on page 47.

## 1.26 A 64-bit Address Space

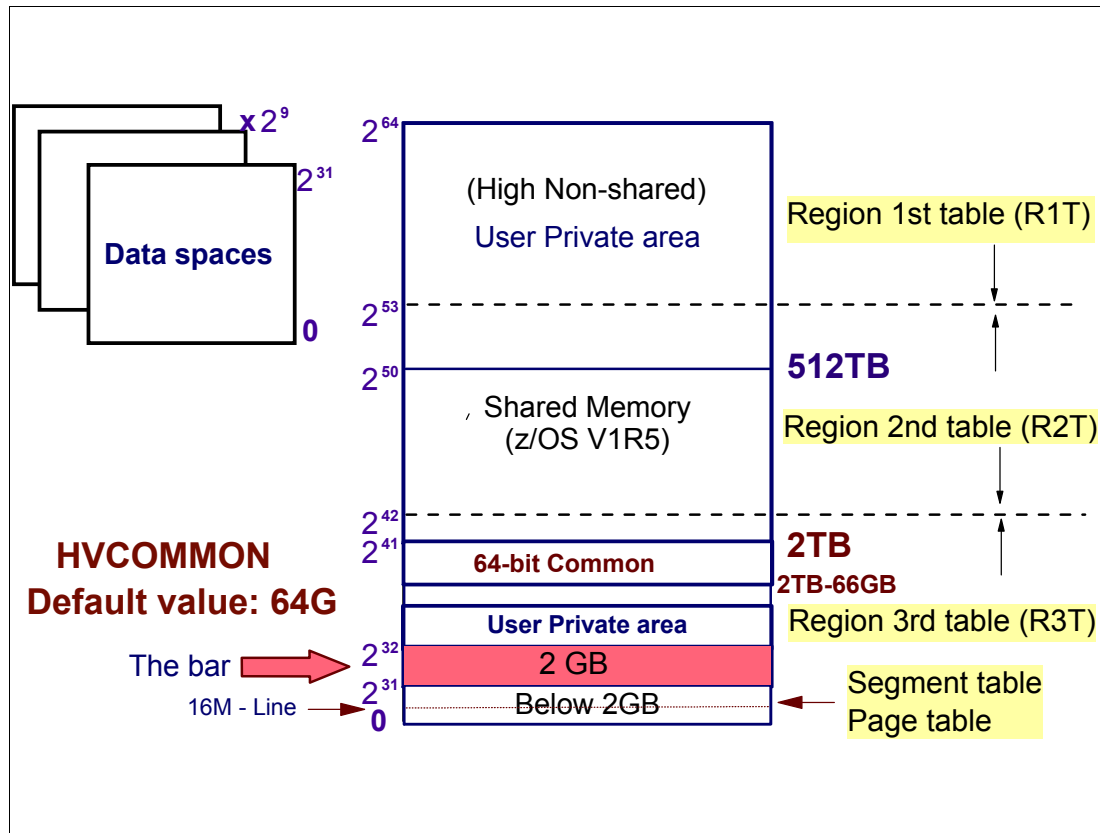


Figure 1-29 The 64-bit address space

### 64-bit virtual address space

The virtual memory 2 GB limit was overcome with z/Architecture, as we already saw. Then, the size of the 64-bit AS is up to 16 EB addresses, which makes the new AS 8 billion times the size of the former S/390 AS. However, programs continue to be loaded and run below 2 GB of addresses; these programs can access data that resides above 4 GB addresses. At z/OS 2.3 is now possible to load programs above the bar (RMODE64), with certain limitations.

On the other hand, z/OS supports for up to 4 TB of CPC memory on a single LP image in a z14 with up to 32 TB total CPC memory.

### The line

This line exists for compatibility reasons. The line is the 16 MB address separating the set of virtual addresses below 16 MB and the ones above 16 MB. Very old programs (AMODE24 and RMODE24) are loaded and access only addresses below this line.

## The bar

For compatibility reasons, the layout of the virtual addresses for an AS is the same below 2 GB, providing an environment that can support both 24-bit and 31-bit addressing. The set of addresses that separates the virtual addresses below the 2 GB and above is called the *bar*, as shown in Figure 1-29 on page 51. The bar is 2 GB thick. As a result, the virtual addresses between 2 GB and 4 GB never will be obtained in the results of a Getmain, for reasons that are outside the scope of this book.

Now, we cover some areas of the 64-bit AS.

## Common area

Common area is a set of virtual addresses located in all address spaces, where the contents are shared by all programs running in all these address spaces and consequently in this z/OS. There are three common areas:

- ▶ Below the line with the components: Nucleus (for z/OS code), SQA (z/OS control blocks), LPA (for subsystem and application code) and CSA (mainly subsystem control blocks).
- ▶ Above the line and below the bar: with components: Extended Nucleus (for z/OS code), Extended SQA (z/OS control blocks), Extended LPA (for subsystem and application code), and Extended CSA mainly subsystem control blocks).
- ▶ Above the bar with the only component High CSA (mainly subsystem control blocks).

In summary, the Nucleus contains the z/OS kernel (CPC memory resident) programs, the LPA contains subsystem and reentrant load modules, SQA contains z/OS control blocks, and the CSA contains subsystem control blocks.

## User private area

Private area is a set of virtual addresses located in an AS, where the contents are only shared by all programs running in this AS.

The *user private area* includes:

- ▶ Low private: The private area below the line
- ▶ Extended private: The private area above the line and below the bar
- ▶ High Non-shared: The private area above the bar and not shared
- ▶ High Shared: The private area above Shared Area

Just for comparison purposes, the common area is a set of virtual addresses located in all address spaces, where the contents are shared by all programs running in all these address spaces. Shared area is a set of virtual addresses located in some address spaces, where the contents are shared by all programs running only in these address spaces. One example of the use of shared area is the Db2 address spaces.

## 1.27 Cross memory

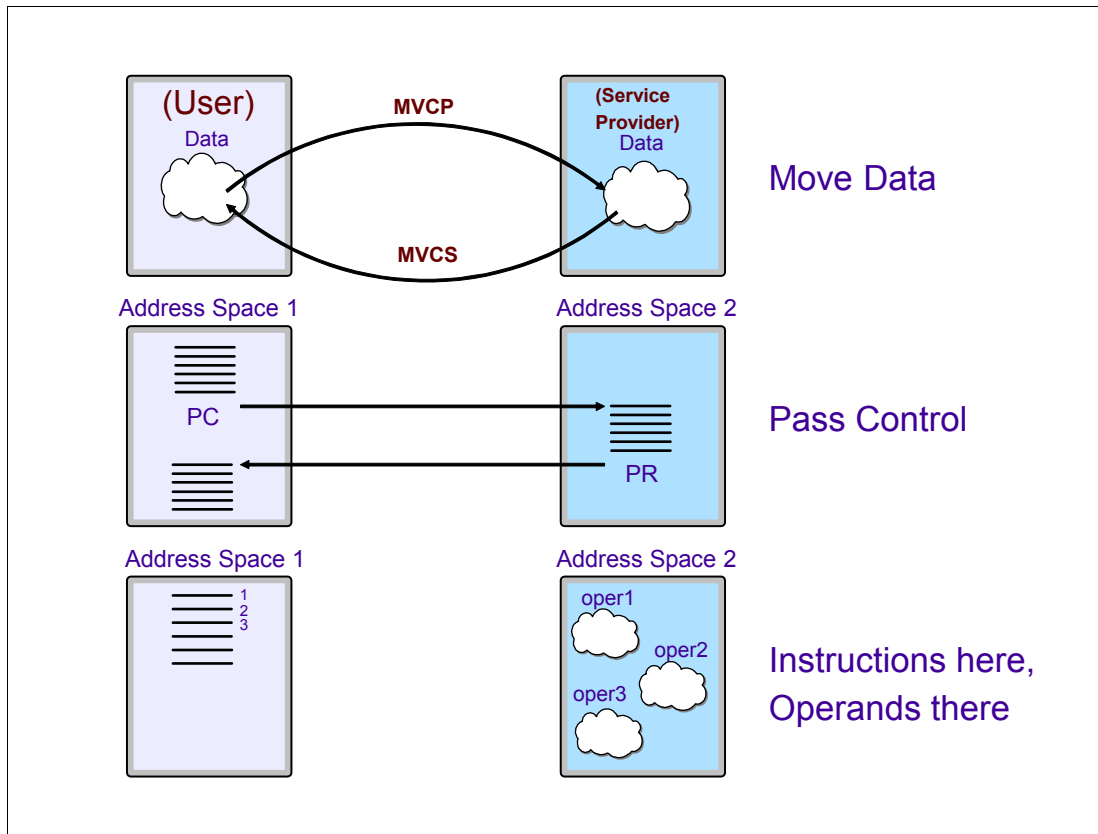


Figure 1-30 Cross memory

### Cross memory

Cross-memory (XM) is an evolution of virtual storage at z/Architecture. It has in reality two major objectives:

- Pass control synchronously between instructions located in distinct address spaces.

As we can see at the Figure 1-30, there is an instruction PROGRAM CALL (PC), able to do that. To return the origin AS, there is the instruction PROGRAM RETURN (PR).

Synchronous cross-memory communication enables two load modules (executing programs) located in private areas of different address spaces to communicate synchronously. For example, cross-memory communication takes place between load modules located at AS 2, which gets control from AS 1, when the PC instruction is issued. Usually the PC-called load module (running in the AS 2) provides a service requested by the caller (at AS 1) and then returns control through the PR instruction. The called PC routine executes under the same task (TCB) as the load module that issues the PC. For instance, address space 1 may be an IBM CICS® address space that invokes address space 2 using Xmemory(PC), such as a data base management system's address space asking for a query operation to be performed.

- Move data synchronously between virtual addresses located in private areas of distinct address spaces. Refer to Figure 1-30.

This can be implemented by the use of the SET SECONDARY ADDRESS REGISTER (SSAR) instruction. It points to an AS and makes it secondary. A secondary AS 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 (MVPU), the above objective can be accomplished.

## 1.28 Access register mode (data spaces)

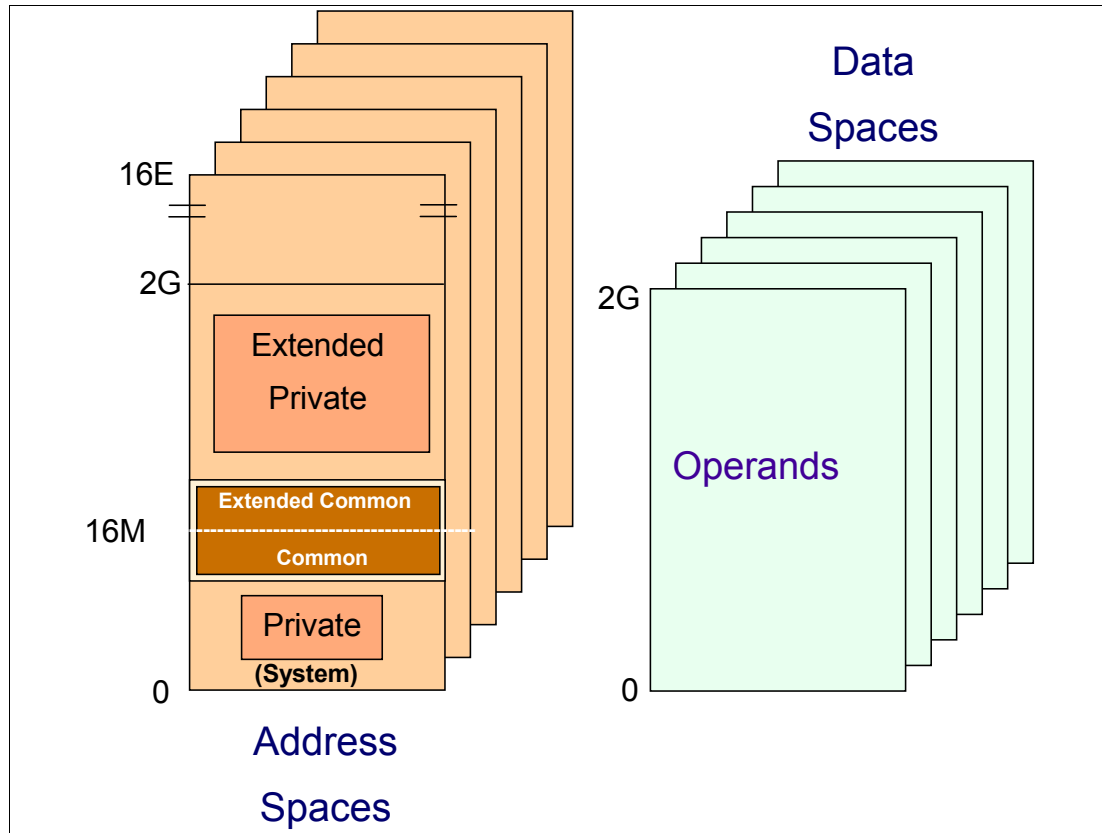


Figure 1-31 Address spaces and data spaces

### Data spaces

Data spaces are data-only (containing no instruction virtual addresses) virtual address spaces that can hold up to 2 GB of addresses. A data space provides integrity and isolation for the data pointed to by its virtual addresses. They are a very flexible solution to problems related to accessing large amounts of data in CPC memory as with Db2 buffer pools. The idea of data spaces comes from researchers, that is, the idea of separating data and instructions into different memory spaces.

As stressed above, a data space contains only virtual addresses of data on which to perform operations (operands), and does not contain virtual addresses for instructions. As an address space, a data space also has a segment table pointing to page tables. Consequently, when accessing data spaces, DAT must be smart enough to manage the following translation activities:

- ▶ Use the AS segment table to translate virtual addresses for instructions.
- ▶ Use the data space segment table to translate operand (data) virtual addresses referred by those instructions.

In order to exploit the data space concept, a program must be executed with the PU in Access Register mode (PSW bit 16 off and bit 17 on). This mode is set through the SST

ADDRESSABILITY CONTROL (SAC) instruction that tells DAT to use another segment table to translate the data (operands) virtual address. In this case, the segment table associated with a data space. The next step is to use an access register to point indirectly to the data space segment table.

### Access registers (ARs)

An *access register* (AR) is a hardware register that a program uses to identify indirectly a data space segment table. Each PU has 16 ARs, numbered 0 through 15.

## 1.29 z/Architecture time facilities

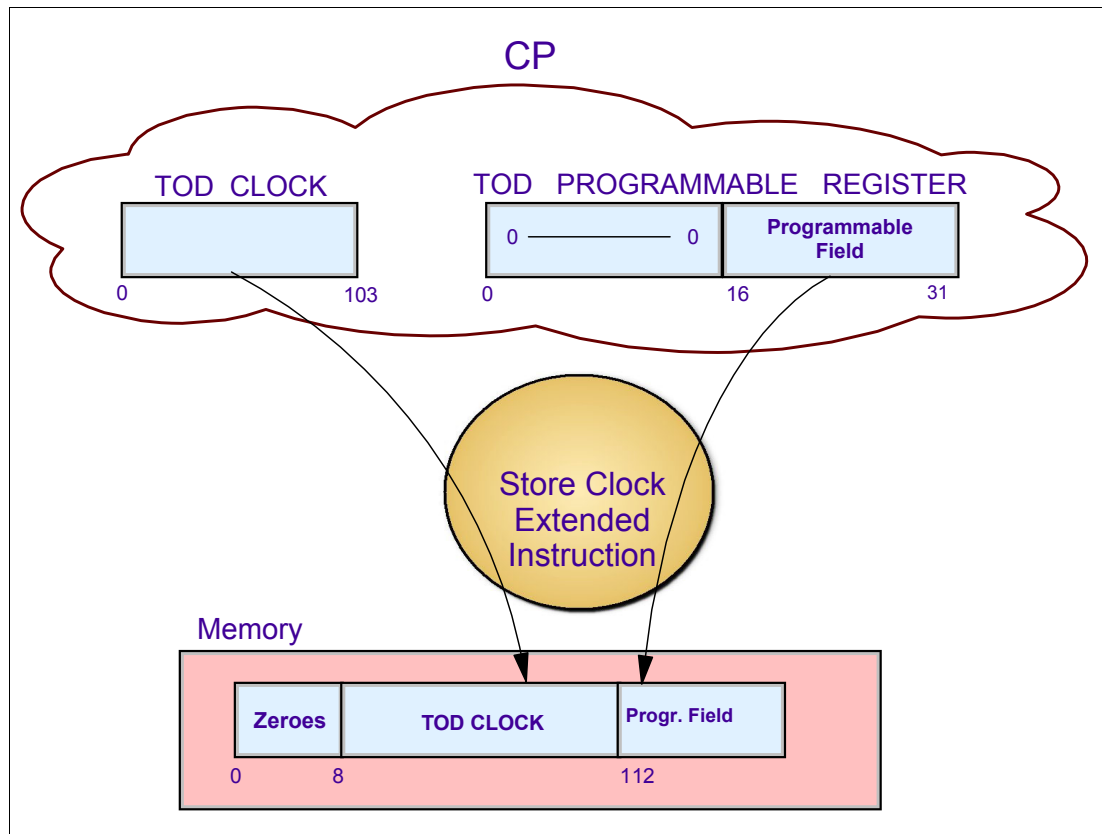


Figure 1-32 Store clock extended instruction

### z/Architecture timing facilities

The a/Architecture timing facilities are the:

- ▶ Time of Day (TOD) clock
- ▶ Clock comparator
- ▶ CPU timer
- ▶ Service Time Protocol (STP)

#### TOD clock

The TOD clock is a 104-bit counter register inside of the z14 CPC Drawer. In a multiprocessing configuration, a single TOD clock is shared by all PUs. The TOD clock provides a high-resolution measure of real time suitable for the indication of date and time of

day. This timing is a precious information source for operating systems, databases, and system logs in a commercial environment.

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 1 in bit 51, every microsecond. In models having a higher or lower resolution, a different bit position is added at such a frequency that the rate of advancing the clock is the same as if 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 isotope 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 PU is in a wait state or whether the PU is in operating, load, stopped, or checkstop states.

z/OS assumes that 01/01/1900 is the zero point for TOD.

### **Instructions for storing the clock**

Two modern instructions are used by z/OS to alter and store its contents in memory:

- ▶ SET CLOCK EXTENDED (SCKE) instruction changes the contents of 104 bits TOD from a memory location. When running under PR/SM, z/OS is not allowed to modify, for any reason, the contents of the TOD clock. This facility falls in the scope of CPC resources, not in the scope of logical partitions. When this happens, the logical PU is intercepted and PR/SM LIC code emulates the modification.
- ▶ STORE CLOCK EXTENDED (STCKE) instruction, as pictured in Figure 1-32 on page 55, moves into memory 104 bits plus the program fields (16 bits) from the TOD programmable-register-fields register. There is one register per PU, and it is used to guarantee the uniqueness of the TOD clock value when stored in CPC memory. The former STCK-instruction stored data will wrap at the first microsecond of the year 2042, and it will be a problem like the Y2K problem in the year 2000. The more obvious solution is to replace this instruction by the SCKE and using some of the programmable bits as an epoch index. Since z/OS version 2.3, the z/OS system started making this migration internally.

### **Clock Comparator**

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, a software application can be alerted when a certain amount of wall clock time has elapsed, or at a specific hour of the day.

### **CPU Timer**

The CPU timer is a binary counter within a PU, with a format that is the same as that of bits 0-63 of the TOD clock, except that bit 0 is considered a sign. The PU timer nominally is decremented by subtracting a 1 in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is decremented at a frequency such that the rate of decrementing the PU timer is the same as if a one were subtracted in bit position 51 every microsecond. The PU timer requests an external interrupt with the interrupt code 1005 hex whenever the PU timer value is negative (bit 0 of the PU timer is one).

The CPU Timer is used by z/OS to account the PU time consumed by all the dispatching units (TCB and SRB) in the system. When a DU is dispatched, a value X is inputted by z/OS at the CPU timer. When the DU is interrupted, a value Y is stored in CPC memory coming from the CPU Timer. X minus Y is the amount of PU time processed along such dispatching interval

and it is accumulated at ASCB or at ENCB (when the DU runs under an enclave) control blocks.

The CPU Timer is accessed by the instructions: STORE CPU TIMER and SET CPU TIMER.

The CPU timer stops when the PU is in stop state. This state may be caused by operator hardware intervention at the Hardware Management Console (HMC), or in PR/SM mode, when a shared logical PU is ready, meaning not executing in a physical PU.

### **External timer reference (ETR)**

There is a long-standing requirement for accurate time and date information in commercial data processing. As single operating systems have been replaced by multiple, coupled operating systems running on multiple CPCs, 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 external time source.

The IBM z/Architecture external time reference (ETR) allows the synchronization of the CPC time-of-day (TOD) clocks to ensure consistent time stamp data across multiple CPCs. Then, ETR provides a mean of synchronizing TOD clocks in different CPCs 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. It is important to note that, the TOD synchronization between CPCs is mandatory for the implementation of a Parallel Sysplex, where the participant z/OS systems are located in distinct CPCs.

### **ETR time**

In defining an architecture to meet z/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 Coordinated Universal Time (UTC) time source.

## 1.30 Server Time Protocol (STP)

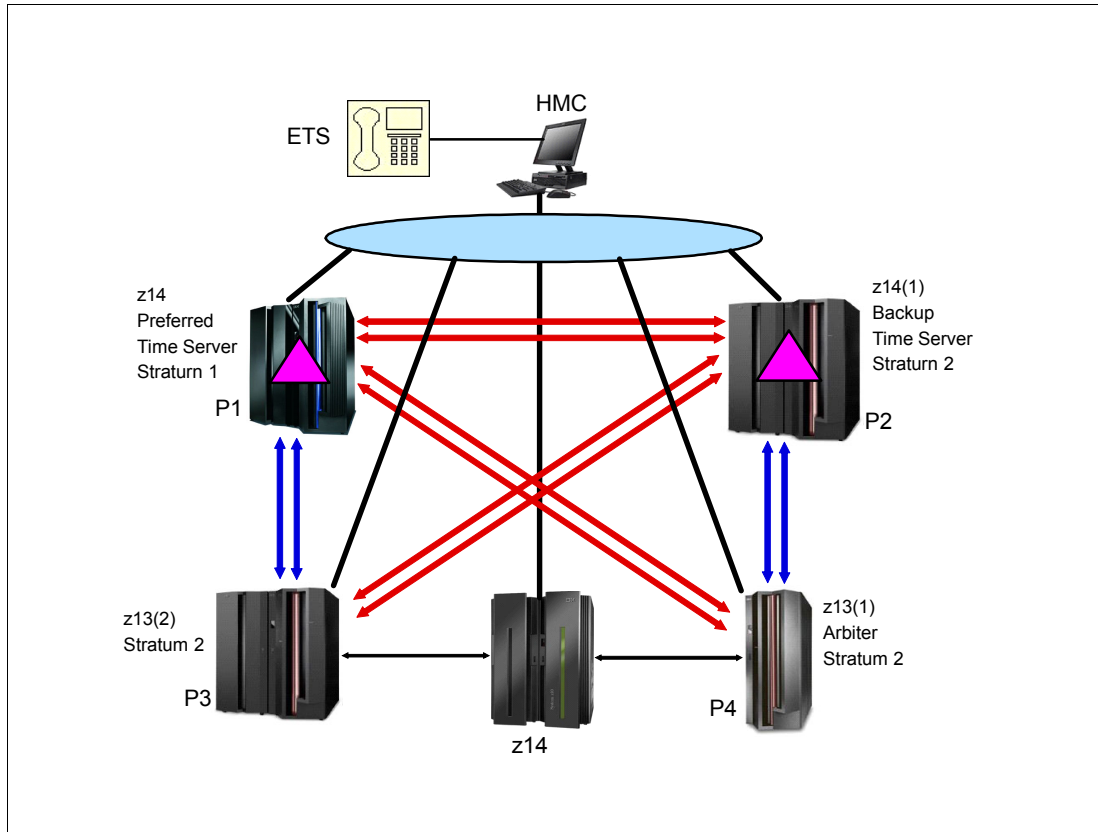


Figure 1-33 Server time protocol full configuration

### Server Time Protocol (STP)

Server Time Protocol (STP) is the modern and cheap way in z/Architecture to synchronize the time of day (TOD) clocks contained in distinct IBM Z family CPCs. STP allows consistent TOD information to be exchanged between z/OS systems and CFCC operating systems running in different CPCs. STP is designed to:

- ▶ Support a multi-site CPC 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 multi-site Parallel Sysplex.
- ▶ 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.

The logic of STP is executed by the SAP PU connected to other CPCs through the same CF Link, the same used in Parallel Sysplex communication, that is, z/OS with coupling facilities.



## 1.31 Hardware Configuration Definition (HCD)

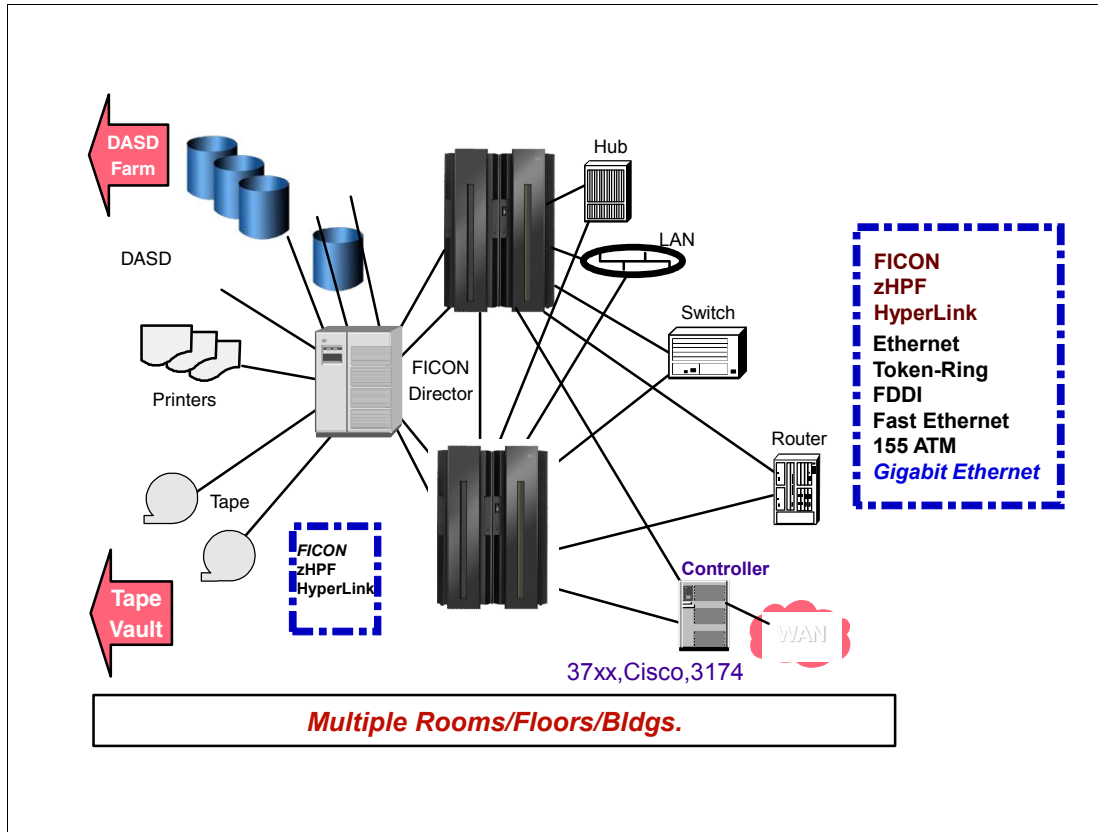


Figure 1-34 Data center hardware elements

### Hardware Configuration Definition (HCD)

You must define the CPC, logical partitions and I/O configuration and some IPL options to z/OS and to SAp and the channel subsystem (I/O channels) through the Hardware Configuration Definition (HCD). HCD is a z/OS component that consolidates these hardware and software definitions under a single interactive TSO end-user interface. The validation checking that HCD does as you enter data helps to eliminate errors before you attempt to use the configuration. The HCD configuration is stored at the z/OS data set named IODF. Using a utility program (IOCP) this configuration is stored at an IOCD file at the hard disk of the z14 support element. At Power On Reset, this configuration is copied to the Hardware System Area (HSA) at the CPC memory start.

As you can imagine, the mainframe platform cannot be plug-and-play, for security reasons.

HCD provides the capability to make both hardware and software I/O configuration changes dynamically. No need for z/OS PL, and no need for logical partition initialization.

The hardware configuration part of the HCD definition describes these resources and the connections between these resources. The resources include:

- ▶ CPCs
- ▶ Logical Channel Subsystems (LCSS)
- ▶ Logical partitions
- ▶ I/O Channels
- ▶ FICON Directors (switches)

- ▶ Controllers
- ▶ Devices such as tape, printers, and DASD

The last four items in the above list are called I/O configuration.

Figure 1-34 shows a typical IBM Z family data center. As you can see, the complex consists of separate I/O devices and networks connected through high-speed data links to the CPC, which comprises PUs (CP, zIIP, IFL), CPC memory, and I/O channels. It shows connections among CPCs, as well. z/Architecture provides up to 1,512 high-speed data buses, called I/O channels, per CPC. Included in those are the OSA channels used to implement network connection.

Recently, IBM introduced a product called zDAC (z/OS Discovery and Autoconfiguration). zDAC is very convenient during partial dynamic changes in a base configuration (such as a new DASD controller), where the modifications are discovered, informed to the installations in order to be modified or not in the base configuration.

## 1.32 Logical channel subsystem (LCSS)

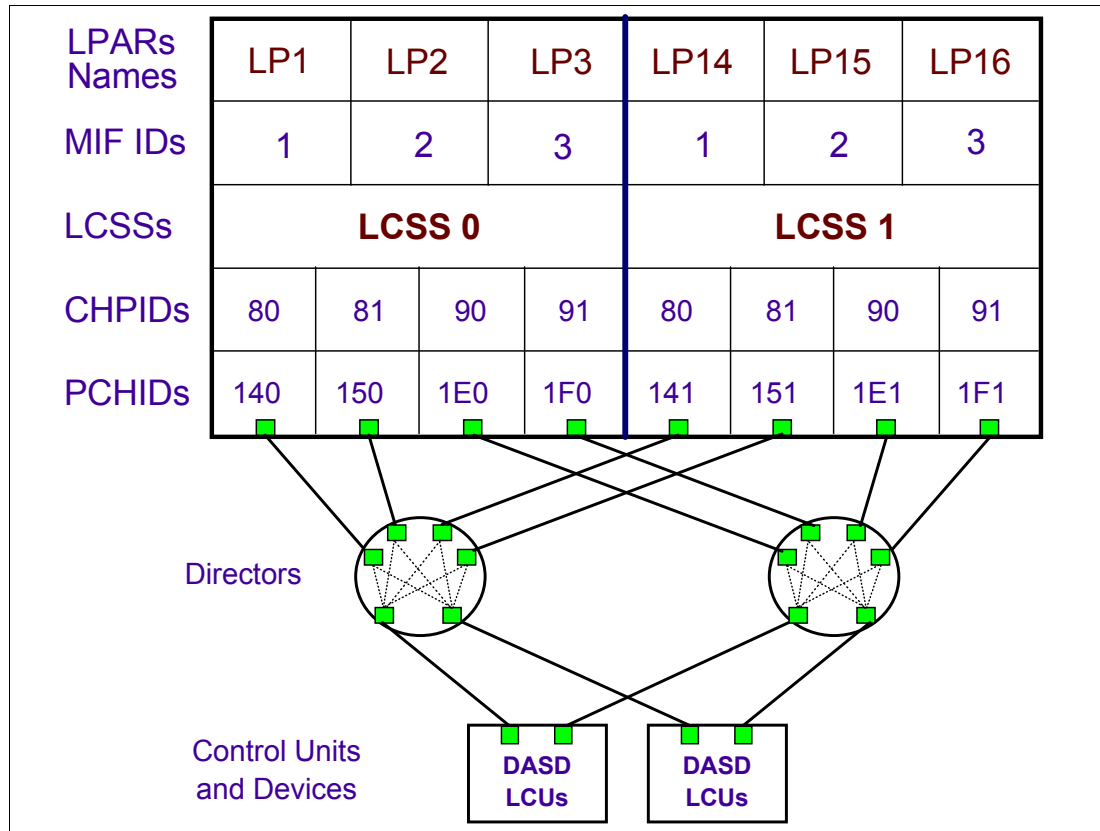


Figure 1-35 A CPC with two LCSS

A logical channel subsystem (LCSS) has several definitions, such as, a set of:

- ▶ 256 I/O channels, or
- ▶ 15 logical partitions in the same CPC, or
- ▶ 4 subchannel sets.

For the moment, we stay with the first definition.

Many IBM manuals mention channel subsystem only, but meaning the same as LCSS. Also, some manuals include the SAPs together with the channels in the LCSS.

The LCSS design provides functionality in the form of multiple LCSS. Up to 6 LCSS can be configured within the same z14, delivering a significant increase in I/O performance and scalability.

The IBM Z family was designed for processing mainly commercial workloads, dominated by a high number of I/O operations. Furthermore, for example, the z14 has a very high processing capacity increasing the burden on the I/O configuration. In support of this, the introduction of several LCSS allows significantly more channels, more logical partitions, and more devices in your IBM Z family I/O configuration.

Previously, before the implementation of multiple LCSS, a maximum of only 256 I/O channels per CPC was possible, which is too low for large customers. These channels were identified by a 4-bit CHPID (256-bit values). Due to compatibility reasons, the CHPID could not be

enlarged. The solution is the creation of several LCSS, and the I/O channel is identified by the CHPID qualified by the LCSS Id. Then, the IBM Z family CPCs provide the ability to define more than 256 CHPIDs in the system through the multiple CSSs. To avoid synonym CHPID I/O channels, a logical partition must belong to just on LCSS, which may contain up to 15 logical partitions.

At the HCD level, logical partitions cannot be added until at least one LCSS has been previously and hierarchically defined. Logical partitions are defined under a LCSS, not directly under the CPC. A logical partition is associated with one LCSS only. CHPID numbers are unique within the logical partitions of a LCSS. However, the same CHPID number can be reused within all LCSSs.

At Figure 1-35 on page 61, we have:

- Two LCSS (0 and 1)
- LP1, LP2 and LP3 belong to the LCSS0 and LP14, LP15 and LP16 belong to LCSS1.
- There are two different I/O channels with the same CHPID, that is, 80. However, they belong to different LCSS, where one is qualified by LCSS0 and the other by the LCSS1. These I/O channels have different PCHIDs (physical channel ID). The I/O channel PCHID identifies the location of the I/O channel in the z14 PCIe I/O drawer. There is just one PCHID per CPC.

### 1.33 Start Subchannel (SSCH) instruction logic

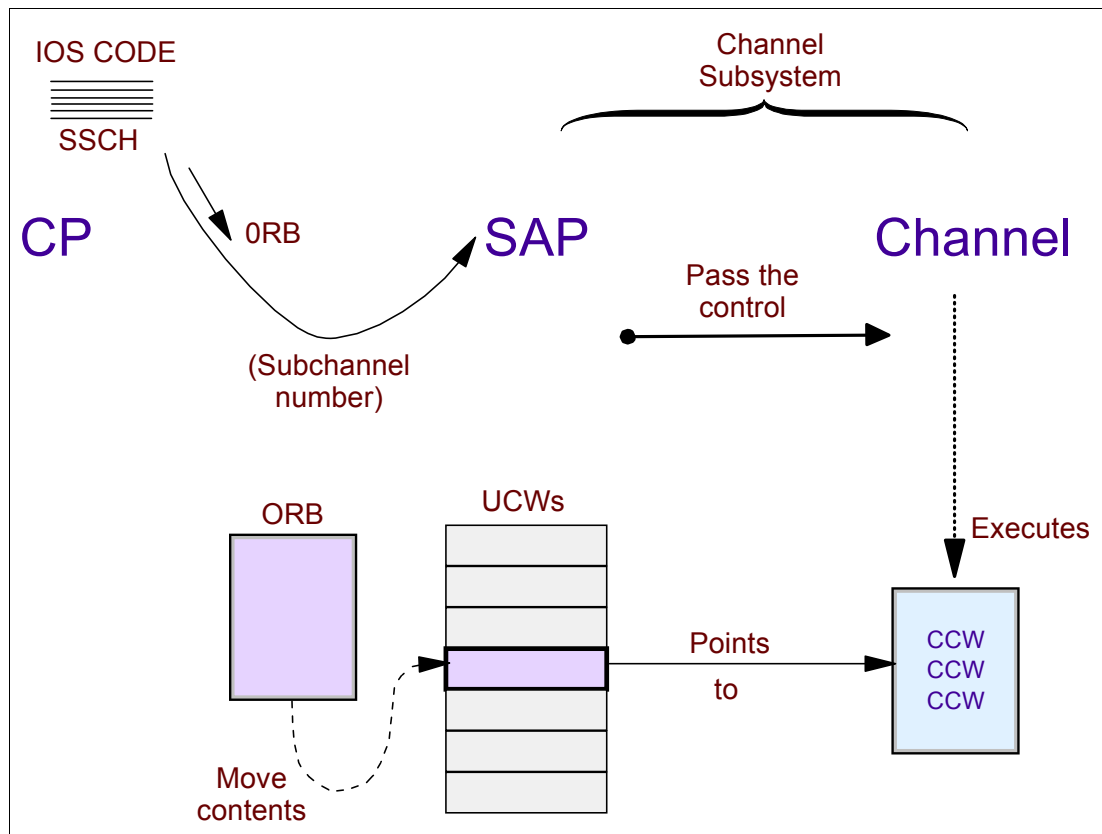


Figure 1-36 SSCH instruction logic

## IOS logic and the 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 program task. Each I/O device at IOS is represented by a Unit Control Block (UCB), as defined by the installation at the HCD. The UCB describes the device and just one running I/O operation.

If the I/O operation target device has its base UCB not busy (busy bit off), there is no other ongoing I/O operation running in the device represented by this base UCB. In this case, IOS issues the SSCH instruction (requesting the I/O operation to SAP) and switches the busy bit at this corresponding base UCB.

This bit will be switched off, at I/O interruption time, to indicate the end of such I/O operation.

However, due to the implementation of the function Parallel Access Volume (PAV), the 3390 device accepts concurrent I/Os. In this case, if the base UCB is busy, IOS may use shared UCBs named aliases, in order to start another I/O operation towards the same 3390 device.

If there are no available UCB aliases, this I/O request is queued at the base UCB level, also called the IOS queue.

## SSCH logic

The SSCH instruction has two operands:

- ▶ General register 1 contains the device identification (where the I/O operation will be executed) using the device subchannel number (refer to “Subchannel and subchannel numbers” on page 72).
- ▶ The second-operand address is a pointer to the operation request block (ORB) that describes the I/O operation. 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 at CPC memory. Refer to *z/Architecture Reference Summary*, SA22-7871, for the ORB contents.

The SAP is signaled to asynchronously perform the start of the I/O operation for the associated device, and the execution parameters that are contained in the designated ORB are placed at the designated subchannel (UCW), as follows:

- ▶ SSCH microcode in the PU moves the ORB contents into the dynamic part of the respective subchannel (UCW) and places this UCW in a specific Hardware System Area (HSA) queue called the *initiative queue*. There is one initiative queue per each SAP. Refer to “Subchannel and subchannel numbers” on page 72, to get more information about the subchannel concept.
- ▶ After that SSCH logic completes, the next IOS instruction is executed, which later will allow the use of the PU in another task. Probably, the requesting task is placed in wait state, waiting for the end of this I/O operation.

## System assisted processor (SAP)

A SAP is a PU that relieves z/OS (and consequently, the PU involvement) during the setup of an I/O operation. It does this I/O operation scheduling, that is, it finds an available FICON channel path to the device and guarantees that the I/O operation really starts. SAP, however, is not in charge of the movement between CPC memory and the FICON I/O channel. At I/O operation end, the FICON I/O channel signals with an I/O interrupt. This interrupt is passed to SAP, by the FICON I/O channel itself. Then, SAP tries to find an enable CP to execute such I/O interrupt.

The z/OS system, the SAP, and the channel subsystem (formed by the FICON I/O channels) need to know the I/O configuration, which contains the following types of information:

- ▶ What devices are attached to each FICON I/O channel.
- ▶ The device types.
- ▶ The device addresses: subchannel number, device number, and device address (also called a unit address). Refer to “Device number” on page 70.

SAP is also in charge of the STP protocol. Refer to “Server Time Protocol (STP)” on page 58.

The z14 has up to 39 dual-thread SAPs in its larger model.

## SAP logic

SAP encounters the UCW in its SAP Initiative queue (as left by the SSCH PU logic) and tries to find an available FICON I/O channel path (channel, port switch, I/O controller and device) that succeeds in starting the requested I/O operation. These initiative queues are ordered by the I/O priority, as determined by the z/OS component Workload Manager (WLM). However, such queue priority order must be installation allowed at HMC.

SAP uses the I/O configuration information described in the HCD, now stored in the static part of the UCW, to determine which FICON I/O channels can reach the target device. Initial selection may be delayed if:

- ▶ Controller interface delays the establishment of the requested I/O operation. This delay is called Command Reply (CMR) delay.
- ▶ The device is busy, due to shared DASD device hardwired Reserve.

During all of these delays, the I/O request is serviced by the SAP without z/OS awareness. 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 PU. All the time between the SSCH and the real start of the I/O operation is timed by SAP and is called *pending time*.

## I/O data transfer mechanisms

Before we start the description of the elements of an I/O configuration, we cover the methods of transferring data between CPC memory and I/O devices, as follows:

- ▶ Traditional I/O data transfer through the use of the START SUBCHANNEL (SSCH) instruction, channel programs made of CCWs, TCW, and I/O interrupts, as the case of FICON I/O channels. A channel program is usually written by an access method, such as VSAM, and tells the FICON I/O channel what is required along an I/O operation like these: read or write, how many bytes, the address at CPC memory, the location of the block data at the media controller, and so on. More recently a channel program is made of one TCW in the zHPF protocol.
- ▶ z14 point-to-point synchronous very fast I/O data transfer for small distances (up to 150 meters) without I/O interrupts and the CP synchronously spinning for the end of the I/O operation, as in the case of PCIe zHyperlink.
- ▶ 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.
- ▶ Parallel Sysplex data exchanged between z/OS and coupling facilities, through CF links.

## I/O Channels

I/O channels are components of the z14 Channel Subsystems (CSS). I/O channels are processors that are much simpler than a PU. They are able to communicate in a dialog with

I/O controllers (CU) and manage the movement of data between CPC memory and the controller. This data movement is called an I/O operation. I/O channels are located in I/O cards in the z14 PCIe I/O drawer. I/O 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 controllers, such as detailed error information

The different types of channels, including the ones connecting the Coupling Facility (CF) are:

- ▶ FICON Express 6+ at z14, able to use FICON or zHPF protocols. It uses optical fiber at an aggregate data rate of 1.6 + GB/sec.
- ▶ z14 HyperLink Express is a direct connect short distance IBM I/O feature designed to work in conjunction with a FICON or zHPF. It dramatically reduces latency by interconnecting the z14 CPC directly to the I/O Bay of the DS8880 DASD controller. zHyperLink improves application response time, cutting I/O sensitive workload response time in half without significant application changes.
- ▶ Open Systems Adapter (OSA) Express 6S, for network connection with the following type of features:
  - 10 Gigabit Ethernet (10 GbE)
  - Gigabit Ethernet (1 GbE)
  - OSA-Express6S 1000BASE-T Ethernet.
- ▶ IBM Integrated Coupling Adapter (ICA SR) for Short Distance connecting z14 and z13 with z/OS and coupling facilities logical partitions.
- ▶ Coupling Express Long Reach (CE LR) for Long Distance connecting z14 and z13 with z/OS and coupling facilities logical partitions.

### **FICON Directors (also known as FICON Switch)**

The switching function of FICON algorithm is handled by a hardware product called FICON Directors, which operationally connect FICON channels and controller 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 FICON topology.

The implementation of FICON Director improves your I/O configuration in the following disciplines: flexibility by easier I/O re-configuration, simplicity by decreasing the number of I/O elements, disaster recovery by allowing a more efficient remote copy operation.

## 1.34 I/O Interrupt processing

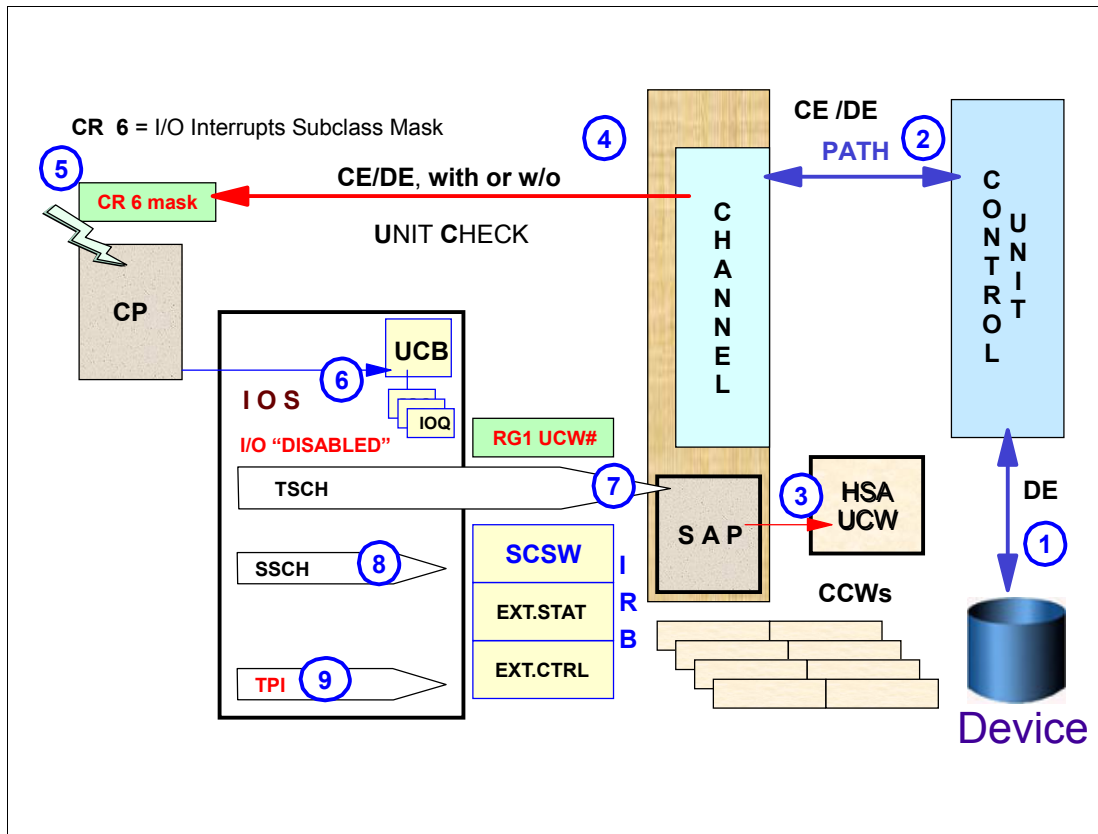


Figure 1-37 I/O interrupt processing

### Device end condition (DE)

A device end status condition generated by the I/O device and presented to the FICON I/O channel following the conclusion of the last step at an I/O operation is a signal of the end of such I/O operation.

### A more detailed view of an I/O interrupt

The traditional I/O for z/Architecture is an interrupt-driven mechanism (the channel interrupts the Cp (not the zIIP) when the I/O completes), as follows:

- ▶ When an interrupt condition is recognized by the channel subsystem, it is indicated at the appropriate subchannel (UCW).
- ▶ The subchannel then has a condition of status I/O interrupt pending.
- ▶ The subchannel becoming status pending causes a FICON I/O channel to generate an I/O-interrupt request to be sent to SAP.
- ▶ An I/O-interrupt request can be brought to the attention of the z/OS only once. An I/O-interrupt request remains pending until it is accepted by any CP in the CPC configuration.
- ▶ When an enabled CP accepts an interrupt request (the CP can be disabled by the PSW bit 6 being off), it saves the current PSW at the PSA I/O old PSW, stores at PSA the



associated interrupt code (describes the device, where the I/O operation finished), and the I/O new PSW is loaded in the current PSW.

The IOS FLIH gains the control and issues the EST SUBCHANNEL instruction that moves to the IRB control block in CPC memory, the contents of the final status located at the subchannel.

### Interruption-response block (IRB)

The IRB is the operand of the TEST SUBCHANNEL (TSCH) instruction. The IRB contains three major fields:

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

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

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

## 1.35 DASD controllers (aka DASD controller )

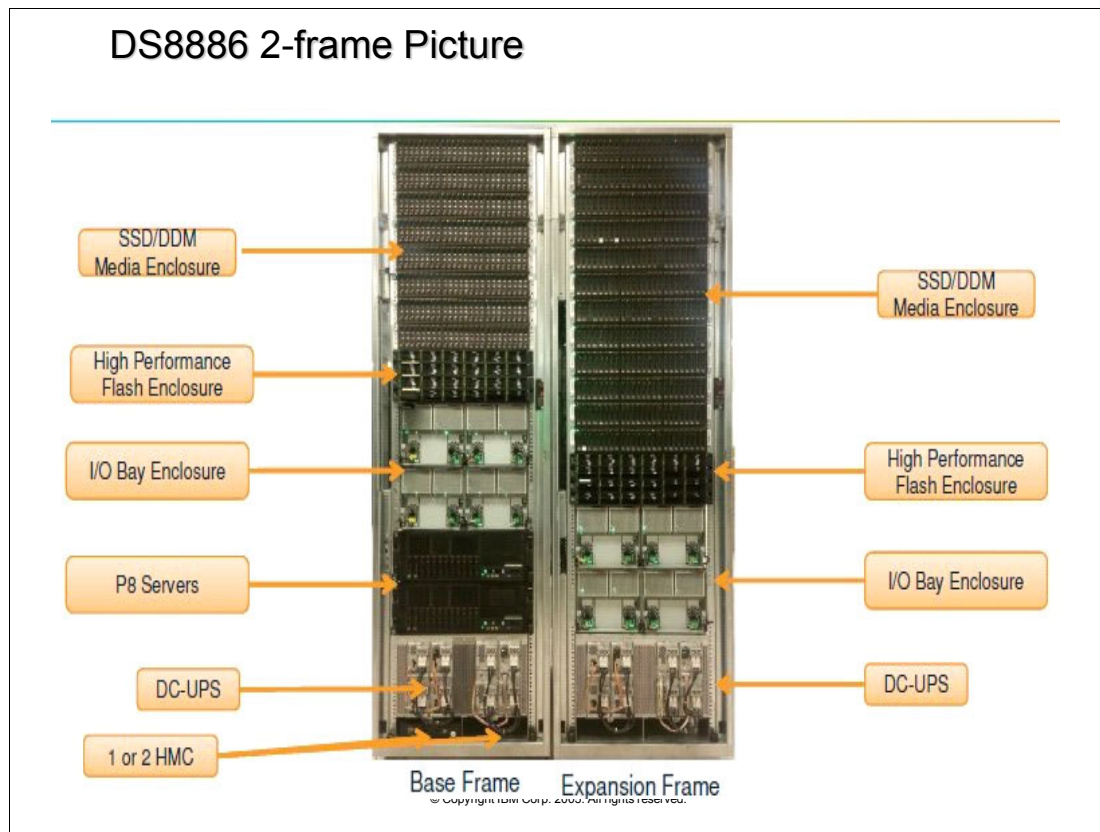


Figure 1-38 DS8886 Storage Controller

## DASD storage 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 DASD controller accepts control signals from the FICON I/O channel, which manages the timing of data transfer over the FICON I/O channel path and provides indications concerning the status of the device. A typical operation is reading a recording medium and recording (storing) data. To accomplish its operations, the device needs detailed signal sequences peculiar to its type of device. The DASD controller decodes the commands received from the FICON I/O channel (as described at the channel program) interprets them for the particular type of device, and provides the signal sequence required for the performance of the operation.

A DASD controller is housed integrated with the I/O devices.

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 (magnetic and flash) in tens of terabyte units

Each DASD controller provides the following capabilities:

- ▶ Caching services to improve performance by minimizing device accesses.
- ▶ Arrays of inexpensive disks (RAID).

Device 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, 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 controllers, named logical control unit (up to 256), each with 256 logical devices.
- ▶ Copy services such as Point-in-time Copy (for non-disruptive backups) and Remote Mirror and Copy (for business continuance).
- ▶ Multiple concurrent types of devices with different technologies (magnetic and flash) 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 I/O channels have a logical view of the devices.
- ▶ Multiple I/O operations in the same logical device (PAV and multiple allegiance).
- ▶ Possibility of hundreds of connecting I/O channel instances.

## Tape Controllers

Tape controllers have some properties of the DASD controllers. Then, the tape controller decodes the commands received from the FICON I/O channel, interprets them for the particular type of device, and provides the signal sequence required for the performance of the operation.

The major difference among both controllers is the media type where data is stored. Tape controllers keep data in cartridges. For example, the 3590 Megstar cartridge has the following properties:

- ▶ 128 or 256 tracks
- ▶ Mechanical speed of 2 m/sec
- ▶ Up to theoretical 20 MB/sec
- ▶ IBMLZ1 data compression (dictionaries)
- ▶ 10 GB (30 GB) or 20GB (60 GB) not compressed data
- ▶ Same external dimensions as former
- ▶ 3480 and 3490 cartridges
- ▶ No interchange with 3480 and 3490 drives
- ▶ Linear serpentine recording
- ▶ New colored inserts identify cartridge type
- ▶ Servo tracks and RAID implementation

Tape technology was nearly dead and buried many times, but it survives. Although still having many opponents, tape technology shows no sign of obsolescence. The major reasons are price and portability (perfect for backups and migrated data sets), and maybe tapes will minimize the big-data tsunami that will come. Some tape media properties:

- ▶ Logical records are grouped in a physical block, in order to have fewer gaps (use the space better) and more efficient I/O. So, there is not Count/Key/Data as in a 3390 track.
- ▶ A set of related blocks is a data set, which maybe multivolume. When a tape data set is deleted the data is still there (no 3390 Erase on Scratch). To reclaim this space the volume must be recycled. A data set can be read forward or optionally backward.
- ▶ At steady state, a tape data rate is faster than magnetic disk media.
- ▶ Tape was first Controller to implement data compression avoiding doing such at the PU, then, saving MSUs.
- ▶ It is possible with z/OS to have blocks with a size up to 1 MB (Large Tape Blocksize).
- ▶ Allows tape device sharing among z/OS systems in a Parallel Sysplex.
- ▶ Minimizes the natural setup performance problems of tapes, such as: mount, load, rewind and data set scan, some solutions were designed:
  - Automatic Robot
  - Virtual Tapes
- ▶ Optimizes tape data management, and many solutions were developed:
  - Cataloging tape data sets in the z/OS catalog
  - To use SMS constructs for tape data sets and volumes
  - Remote Copy (PPRC) to include the data stored in tapes as participant of a DR implementation.
- ▶ Improves data security tape storage including:
  - Data can be encrypted (after compression)
  - Supports WORM (write once, read many) tape data cartridges whose data cannot be altered.

- RACF protection.
- ▶ DFSMSrmm provides tape management functions including validation, overwrite prevention, return to scratch, movement control between libraries and locations, recycling and others.

### Tape Virtualization

By Tape Virtualization (IBM VTS product) we understand, the use of intermediate DASD (named cache) media to store tape data sets in a totally compatible software access.

Then, Tape Virtualization is a hardware-based solution that addresses not only tape cartridge utilization but also tape device utilization and hence overall tape subsystem costs. Because of its transparency to the host software, this solution is readily supported and easy to use.

Three major reasons may justify the tape virtualization concept:

- ▶ Very low occupancy of the tape media
- ▶ Very low utilization of tape devices, including their FICON I/O channels.
- ▶ Tape performance aspects of a tape volume setup, such as: Mount, Load, Rewind.

## 1.36 Device number

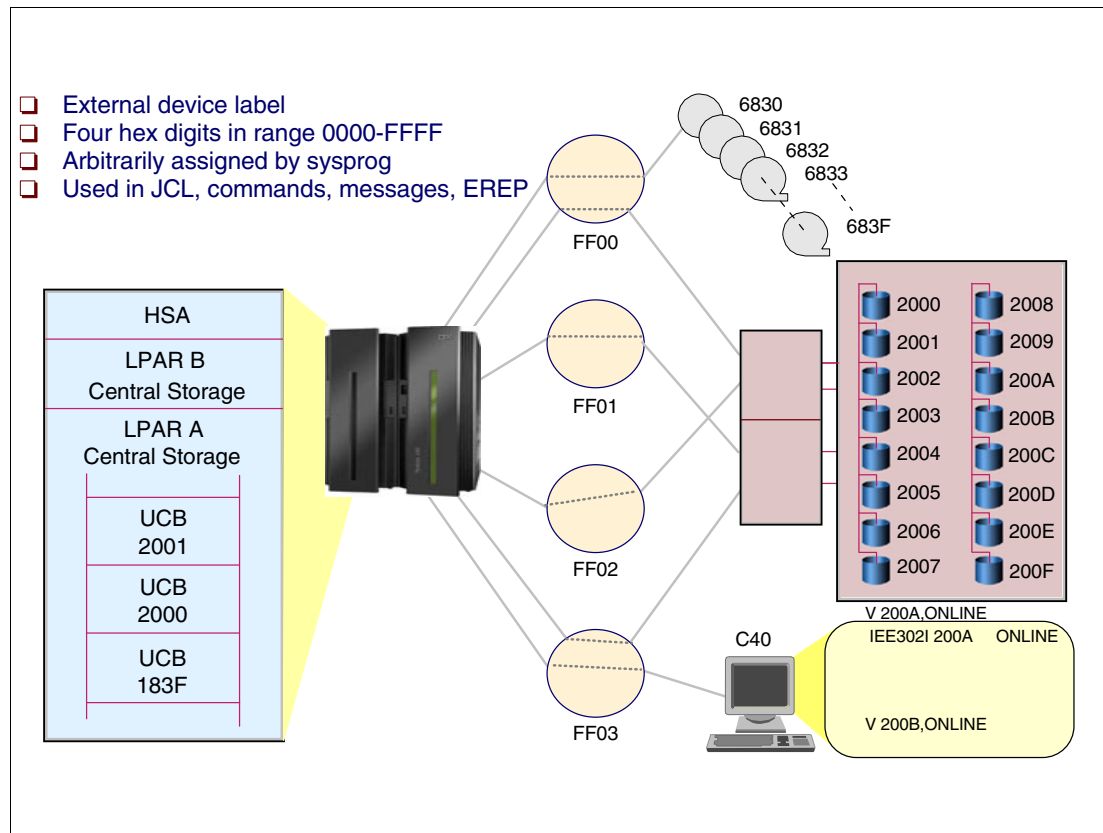


Figure 1-39 Device Number

## I/O devices

An I/O device provides external memory, and a means of communication with such a device and the CPC memory. An I/O device is attached to a controller and it is accessible through an I/O channel path.

Then, an I/O device is the endpoint in the “conduit” between a CPC memory and external data. The major type of I/O devices attached to Z Family CPCs are DASD, tape, and printers.

Each I/O device is represented within IOS by a Unit Control Block (UCB). An UCB holds information about an I/O device, such as:

- ▶ State information for the device
- ▶ Features of the device

Each I/O device is represented within SAP and FICON I/O channels by a Unit Control Word (UCW) located at HSA (also known as subchannel).

An individual I/O device may be accessible to the channel subsystem by as many as eight different FICON I/O channels. Each I/O device is identified by three types of addresses: device number, subchannel number and unit address.

## Device number

Each I/O device is identified by a system-unique parameter called the *device number*. The device number is a 16-bit value that is assigned at HCD allowing a maximum of 65,536 devices. It is a sort of nickname to identify the I/O device during communication between people and z/OS. Refer to Figure 1-39.

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 after the IPL completes can toggle a device online or offline, as follows:

```
V 200A, onl ine
```

The device number is stored in the UCW (in the Hardware System Area - HSA) at Power-on Reset (POR) coming from the IOCDS. It is also stored at UCB at IPL.

## 1.37 Subchannel and subchannel numbers

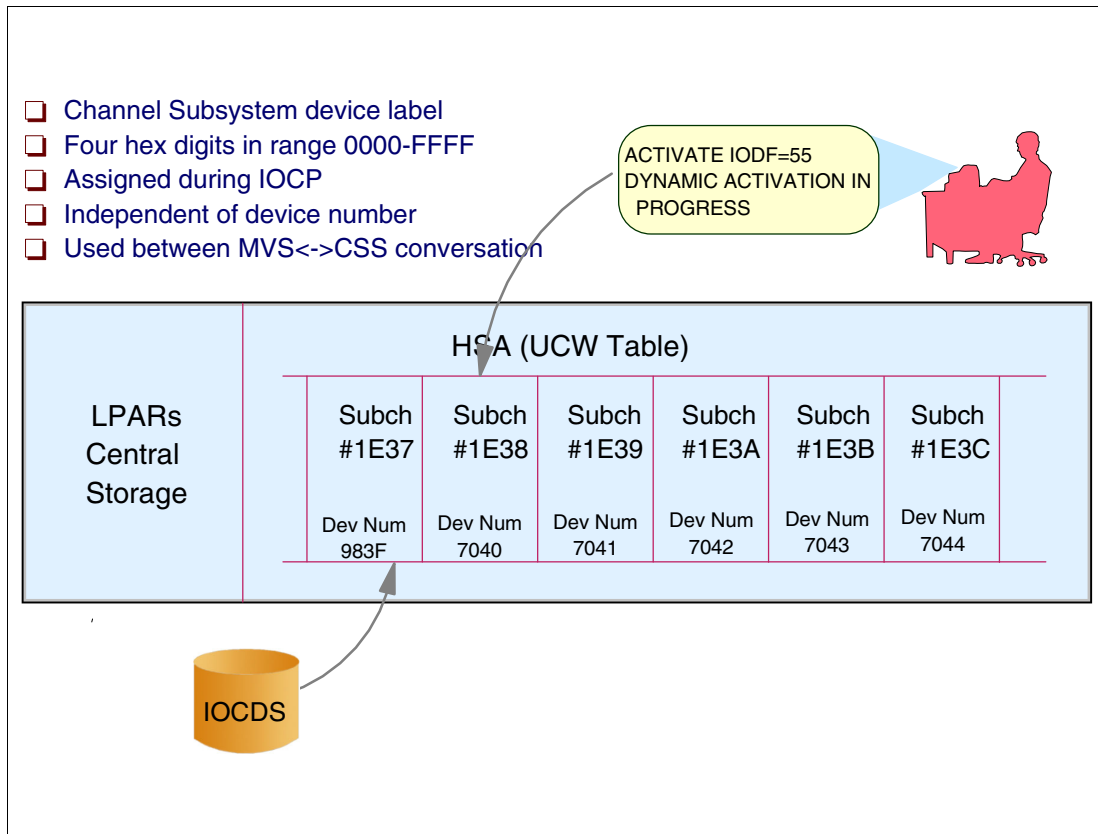


Figure 1-40 Subchannel numbers

### Subchannel

Before covering the concept of subchannel number let's review the subchannel concept.

Subchannel is a z/Architected entity, represented in the HSA by a control block named Unit Control word (UCW). Each I/O device accessible to the channel subsystem is assigned a dedicated subchannel at installation POR time.

Then a subchannel 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 (such as SSCH) that designate the subchannel associated with the device.

Each subchannel provides information concerning the associated I/O device and its attachment to the channel subsystem, and other functions involving the associated I/O device. The subchannel also provides information concerning a current I/O operation, but only one.

Then, the subchannel consists of internal memory (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 UCWs are entries in special index tables named subchannel set or UCW tables. In z14, there are four subchannel sets per Logical Channel Subsystem (LCSS).

### Subchannel number

Subchannel numbers are used to identify an I/O device along the communication between the IOS (through up to 8 instructions, such the SSCH) and the group formed by SAP and the FICON channels.

Then, a *subchannel number* is a 16-bit (2 bytes) value whose valid range is 0000-FFFF and which is used to address a subchannel. In order to address the limitation of up to 64 KB devices at large customers, z14 introduced four subchannel sets per LCSS.

In reality, a subchannel number is the UCW index in the subchannel set table. Refer to Figure 1-40 on page 72, that 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. Refer to “Unit address and logical control unit” on page 74.

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.

Then, there is a maximum of 65,536 subchannels, (64 K) devices per subchannel set and there are 4 subchannel sets per LCSS and there is 6 LCSS per z14.

About the 4 subchannel sets, we may say that the subchannel sets 1, 2, and 3 are used for PAV alias addresses and the secondary devices at remote copy synchronous (PPRC). The subchannel set 0 still reserves 64 KB subchannels for z/OS use.

In z14, the maximum number of UCWs is 64 KB times 6 times 4, because there are four UCW tables per LCSS in the HSA. The same subchannel number is duplicated in different LCSSs. However, it does not pose a problem because the same z/OS can be in just one logical partition that belongs to just one LCSS.

The subchannel number is stored in the UCB at IPL.

**Note:** When you have devices in controller s with the capability of doing parallel I/O operations (Parallel Access Volume - PAV) as IBM DS8000®, you need to define a UCW for each additional parallel I/O to the same device. You must do this because each UCW supports only one I/O operation at a time.

## 1.38 Unit address and logical control unit

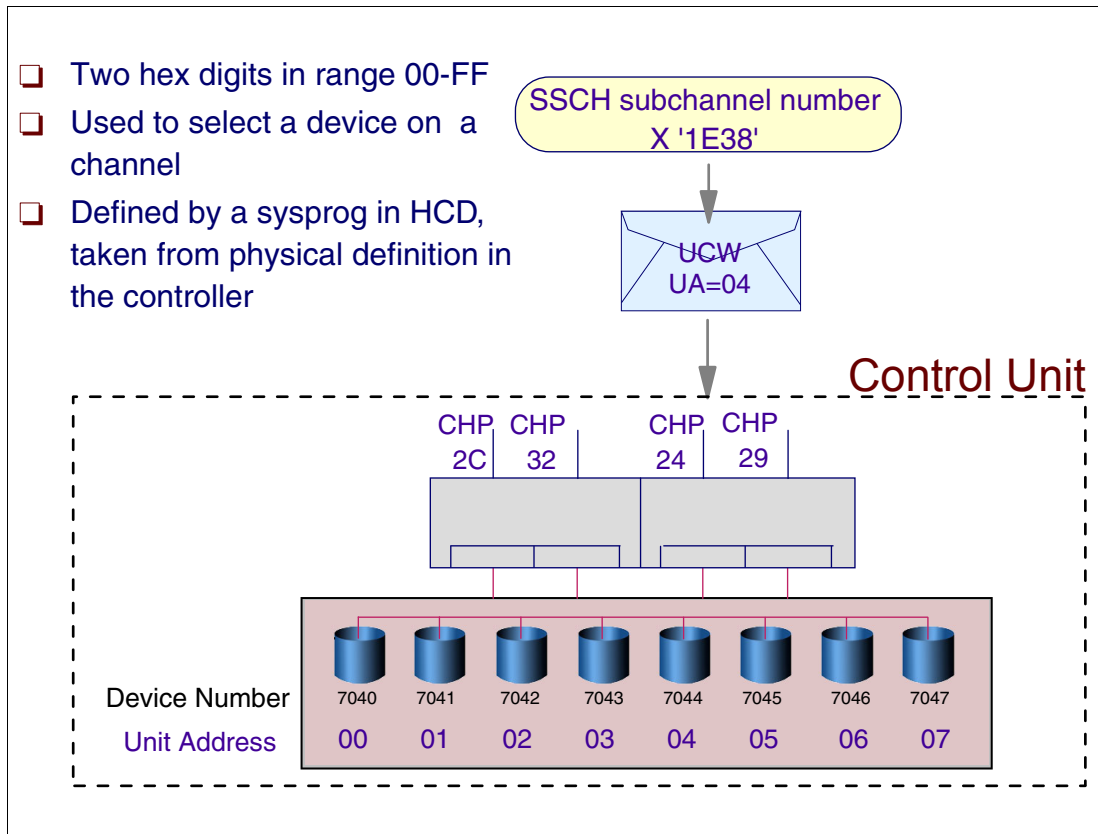


Figure 1-41 Device number, Subchannel number, and Unit address

### Unit address

The *unit address (UA)* or *device address* is used to reference the device during the communication between a FICON I/O channel and the controller 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. Then, it is transmitted over the I/O interface to the controllers by the FICON channel.

### Logical control unit

The UA has no relationship to the device number or the subchannel number for a device, although Figure 1-41 shows the device numbers at the controller. Some controller attached to z14 CPC requires a unit address range starting at X'00', as shown in Figure 1-41. The unit address defined in HCD must match the unit address on the controller where the device has been physically installed by the hardware service representative. To avoid the limitation of only 256 devices per controller, the concept of logical control unit was introduced. You should note that the unit address field cannot be made larger, due to compatibility issues. Then, the solution is to have a physical controller pretending that it is several controllers, each one supporting 256 devices. This simulated set of controllers is called a logical control unit. Then, during the I/O operation, the FICON I/O channel informs the physical controller of these details: the unit address, and the logical control unit number that contains such a device.

### I/O operation

In the Figure 1-41 on page 74, IOS issues the SSCH instruction pointing to the UCW with the subchannel number X'1E38'. One of the FICON I/O channels (CHPID 24, for example),



handling the I/O operation, fetches from the UCW the UA 04. Channel 24 starts the dialog with the controller 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 z/OS.
- ▶ The subchannel number is used in communication between PU (z/OS) and channel subsystem.
- ▶ The unit address is used in communication between the channel and the controller.

### DISPLAY M command

Use the **DISPLAY M** command to display the status of sides, servers, channel paths, devices, CPC memory, and expanded memory, 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 on page 75, the command output shows the device status as online and for each of the CHPIDs, (10, B0, 88, and DC), whether the channel path is online, the CHPID is online, and the path is operational.

## 1.39 I/O summary

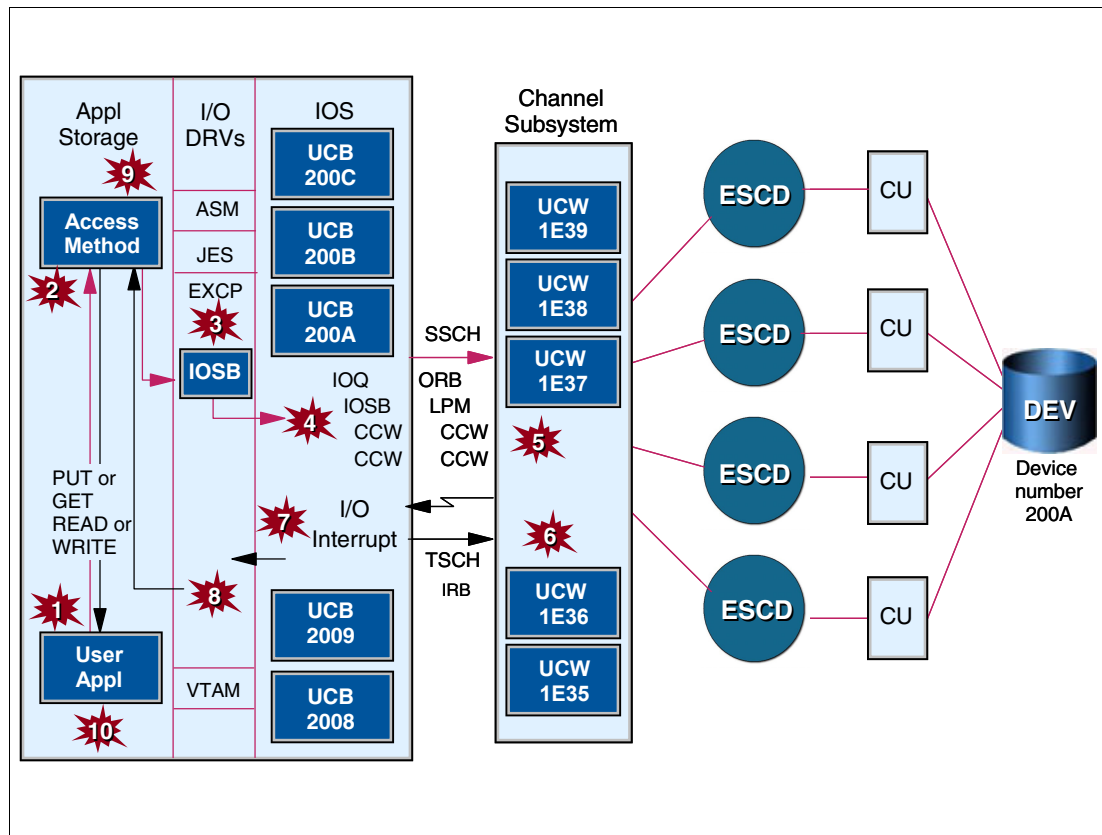


Figure 1-42 I/O summary

### I/O summary

Figure 1-42 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 19 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 distinct 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 Media Manager driver) by issuing the EXCP macro, which expands into an SVC 00 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 PU 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 and there are no available alias UCBS, the I/O request is queued in the UCB control block, following the I/O priority as defined by WLM.

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, controller, and CPC memory.

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

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.

### **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 memory 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.



# Introducing the IBM z14

This chapter introduces the IBM z14.

## 2.1 Introducing the IBM z14

Without question, trust is the most crucial component of successful business interactions. At the core of all business relationships, trust starts with the reliable data and transactions. For that reason, they must always be protected, always available, always delivered with speed, and infused with value.

In addition, no business or organization can ignore the effects of today's digital transformation. Data and transaction volumes are growing exponentially, and workload complexity is rapidly expanding. There is a broad demand for new services and individualized customer experiences. Across industries, newcomers are disrupting markets by using cloud technologies to develop and deploy applications with speed and agility. Established business models must adapt to compete with these new players who are not constrained by legacy and tradition.

The latest member of the IBM Z family, the z14, is built upon a tried and true architecture to support your digital transformation, create a strong cloud infrastructure, and expose back-end services through secure APIs.

The z14 can help your organization make consistently optimal decisions, gain operational data insights so you get the most value from your IT investment, and fully protect your data with pervasive encryption (in-flight and at-rest), while facilitating regulatory compliance.

Leadership for a trust economy can be built on the z14. It is the premier system for enabling data as the new security perimeter, and is designed for data serving in a cognitive era. The z14, more than any other platform, offers a high-value architecture that supports an open and connected world.

## 2.2 The z14

The z14, which is shown in Figure 2-1 on page 79, is the successor of the IBM z13®. It has a newly designed 10 core chip, clocking at 5.2 GHz.

The z14 goes beyond previous designs while continuing to enhance the traditional mainframe qualities, delivering unprecedented performance and capacity growth. The z14 introduces a paradigm shift for protecting data and transactions, from selective encryption to pervasive encryption.

The z14 can be configured with up to 170 processors and up to 30 TBs memory. It offers hot pluggable I/O drawers that build on previous IBM Z innovations, and continues the utilization of advanced technologies such as InfiniBand and RoCE (RDMA over Converged Ethernet).

The z14 has two options concerned with cooling: water and air. You, the customer chooses which configuration is best suited to your environment.

It continues to run all the mainframe operating systems as such: z/OS, z/VM, z/VSE, Linux for z, z/TPF and CFCC (for coupling facility).



Figure 2-1 The IBM z14

## 2.3 z14 Technical Highlights

Below is a summary of the technical highlights:

- ▶ Processor and memory
- ▶ Cryptographic functions
- ▶ Capacity and Performance
- ▶ Virtualization
- ▶ I/O subsystem and I/O features
- ▶ Reliability, availability and serviceability (RAS) design

### **z14 Processor and Memory**

IBM continues its technology leadership with the z14 server. The z14 server is built using the same modular design as the z13. A multi-drawer design that supports 1 - 4 processor drawers per Central Processor Complex (CPC or server). Each processor drawer contains either 5 or 6 Central Processor (CP) single-chip-modules (SCM) and one Storage Controller (SC) SCM. Each CP SCM has ten processor units (PUs or Cores). In addition to SCMs, CPC drawers house the memory DIMMS and connectors for the I/O drawers.

The z14 provides up to a 30% increase in total system capacity over the previous model (IBM z13), and has up to 3.2 times the available central storage (up from 10 TB to 32 TB). The maximum number of processor units (PUs) has increased from 141 to 170.

All z14 (machine type designation 3906) models ship with two frames:

The A-Frame: houses up to four Processor drawers, central storage, a PCIe I/O drawer (Peripheral Component Interconnect® Express) and two 1U servers (Support Elements)

The Z-Frame: houses up to four PCIe I/O drawers, and the optional Internal Battery Facility (IBF)

The PCIe I/O drawers contain I/O cards with I/O processors, known as *channels*.

## **Cryptographic functions**

The z14 provides two major cryptographic functions: CPACF and Crypto Express6S.

The Central Processor Assist for Cryptographic Function (CPACF), standard on every core, supports pervasive encryption and provides hardware acceleration for encryption operations.

The Crypto Express6S feature provides for high-performance cryptographic operations and support for up to 85 domains.

Combined, these two enhancements perform encryption more efficiently on the z14 than on earlier Z platforms.

### ***CPACF***

CPACF is a high performance, low latency co-processor that performs symmetric key encryption and calculates message digests (hashes) in hardware and is the key to Pervasive encryption. On-chip encryption rates on the z14 CPACF are up to 6X faster than the z13.

IBM Z pervasive encryption provides the comprehensive data protection that your organization and customers demand. By placing the security controls on the data itself, the solution creates an envelope of protection around the data. For example, Z pervasive encryption helps protect the at-rest and in-flight data that is on your Z infrastructure. Also, centralized, policy-based data encryption controls significantly reduce the costs that are associated with data security and regulatory compliance. This leap in performance, together with having one dedicated cryptographic co=processor per PU (core) is the key to achieving pervasive encryption.

### ***Crypto Express6S***

The tamper-sensing and tamper-responding Crypto Express6S features provide acceleration for high-performance cryptographic operations and support up to 85 domains. This specialized hardware performs AES, DES/TDES, RSA, Elliptic Curve (ECC), SHA-1, and SHA-2, and other cryptographic operations. It supports specialized high-level cryptographic APIs and functions, including those required in the banking industry.

Crypto Express6S features are designed to meet the FIPS 140-2 Level 4 and PCI HSM security requirements for hardware security modules.

The z14 offers twice the AES performance as the z13, a True Random Number Generator, SHA3 support, and RSA/ECC acceleration.

## **z14 Capacity and Performance**

The z14 server provides increased processing and enhanced I/O capabilities over its predecessor, the z13 system. This capacity is achieved by increasing the performance of the individual PUs (processor units), increasing the number of PUs per system, redesigning the system cache, increasing the amount of memory, and introducing new I/O technologies.

The z14 server is offered in five models, with 1-170 configurable PUs. Models M01, M02, M03 and M04 have up to 41 PUs per CPC drawer, while the high-capacity model, M05, has four CPC drawers with 49 PUs per drawer. The Model M05 is estimated to provide up to 35% more total system capacity than the equivalent z13 Model NE1, with the same amount of memory and power requirements.

With up to 32 TB of memory and enhanced SMT2 (Simultaneous Multi-Threading - two threads per core), the performance of the z14 servers delivers considerable improvement. Uniprocessor performance has also increased significantly - a z14 Model 701 offers average performance improvements of 10% over the z13 Model 701 (observed performance increases vary depending on the workload types).

The IFL, zIIP processor units on the z14 server can be configured to run two simultaneous threads per clock cycle in a single processor (SMT), increasing the capacity of these processors with 25% in average over processors running single thread. SMT is also enabled by default on SAPs.

## **z14 Virtualization**

The IBM z14 server supports z/Architecture mode only and can be initialized either in LPAR mode (sometimes known as PR/SM) or DPM (Dynamic Partition Manager) mode. Only one mode can be active and a Power-on Rest is required to switch between modes.

### ***LPAR Mode (PR/SM)***

LPAR is the traditional mainframe configuration mode. LPAR (or PR/SM) is Licensed Internal Code (LIC) that manages and virtualizes all the installed and enabled system resources as a single large symmetric multiprocessor (SMP) system. This virtualization enables full sharing of the installed resources with high security and efficiency.

LPAR supports configuring up to 85 LPARs, each of which has logical processors, memory, and I/O resources. Resources of these LPARs are assigned from the installed CPC drawers and features. LPAR configurations can be dynamically adjusted to optimize the LPAR or servers' workloads.

For more information about LPAR and PR/SM functions, see Chapter 4, "Virtualization and Logical Partition (PR/SM) concepts" on page 113.

### ***Dynamic Partition Manager mode***

DPM is a new administrative mode (graphical style front-end to PR/SM), introduced for Linux only systems, and available on the IBM z14, IBM z13s®, IBM z13, and IBM LinuxONE™ servers. (A system POR is required to switch between modes).

It is intended to simplify virtualization management and is easy to use, especially designed for those who have little or no experience with IBM Z. It does not require you to learn complex syntax or command structures.

DPM provides simplified hardware and virtual infrastructure management, including partition lifecycle and integrated dynamic I/O and PCIe functions management for Linux running in an LPAR, under KVM on z, and under z/VM 6.4. Using DPM, an environment can be created, provisioned, modified without disrupting running workloads, and monitored for troubleshooting.

DPM provides the following capabilities through the HMC:

- ▶ Create and provision an environment, including new partitions, assignment of processors and memory, and configuration of I/O adapters

- ▶ Manage the environment, including the ability to modify system resources without disrupting workloads
- ▶ Monitor and troubleshoot the environment to identify system events that might lead to degradation

Enhancements to DPM on the z14, simplify the installation of the Linux operating system, support additional hardware cards, and enable base cloud provisioning through OpenStack, including the following enhancements:

- ▶ Support for auto configuration of devices to simplify Linux Operating System Installation, where Linux distribution installers exploit function
- ▶ Secure FTP through HMC for booting and installing an Operating system by using FTP

### **z14 I/O Subsystem and I/O Features**

The z14 server supports both PCIe I/O and InfiniBand I/O infrastructure. PCIe I/O features are installed in PCIe I/O drawers. Up to five PCIe I/O drawers per z14 server are supported, providing space for up to 160 PCIe I/O features.

The system I/O buses take advantage of the Peripheral Component Interconnect Express (PCIe) technology and the InfiniBand technology, which are also used in coupling links.

The z14 connectivity supports the following I/O or special purpose features:

- ▶ Storage connectivity:
  - Fibre Channel connection (IBM FICON):
    - FICON Express16S+ 10 km long wavelength (LX) and short wavelength (SX)
    - FICON Express16S 10 km LX and SX
    - FICON Express8S 10 km LX and SX
  - IBM zHyperLink Express
- ▶ Network Connectivity:
  - Open Systems Adapter (OSA):
    - OSA-Express6S 10 GbE long reach (LR) and short reach (SR)
    - OSA-Express6S GbE LX and SX
    - OSA-Express6S 1000BASE-T Ethernet
    - OSA-Express5S 10 GbE LR and SR
    - OSA-Express5S GbE LX and SX
    - OSA-Express5S 1000BASE-T Ethernet
  - IBM HiperSockets
  - Shared Memory Communication - Remote Direct Memory Access (SMC-R):
    - 10 GbE RoCE (RDMA over Converged Ethernet) Express2
    - 10 GbE RoCE Express
  - Shared Memory Communication - Direct Memory Access (SMC-D) through Internal Shared Memory (ISM)
- ▶ Coupling and Server Time Protocol connectivity:
  - Internal Coupling (IC) links
  - Integrated Coupling Adapter Short Reach (ICA SR)
  - Coupling Express Long Reach (CE LR)
  - HCA3-O, 12x Parallel Sysplex InfiniBand (IFB) coupling links
  - HCA3-O, 1x Parallel Sysplex InfiniBand (IFB) coupling links



- ▶ Cryptography:
  - Crypto Express6S
  - Crypto Express5S
  - Regional Crypto Enablement
- ▶ IBM zEnterprise® Data Compression (zEDC) Express feature

For further information on IBM Z connectivity see Chapter 3, “IBM Z connectivity” on page 101, or *IBM Z Connectivity Handbook*, SG24-5444.

## **z14 Reliability, availability and serviceability (RAS)**

System reliability, availability, and serviceability (RAS) is an area of continuous IBM focus and a defining IBM Z platform characteristic. The RAS objective is to reduce, or eliminate if possible, all sources of planned and unplanned outages, while providing adequate service information if an incident/outage does occur. Adequate service information is required to determine the cause of an issue without the need to reproduce the context of an event.

All IBM Z servers are designed to enable the highest availability and lowest downtime. Comprehensive, multi-layered strategy includes:

- ▶ Error Prevention
- ▶ Error Detection and Correction
- ▶ Error Recovery

With a properly configured z14 server, further reduction of outages can be attained through First Failure Data Capture (FFDC) - designed to reduce service times and avoid subsequent errors, improved non-disruptive replace, repair, and upgrade functions for memory, drawers, and I/O adapters. In addition, z14 servers have extended non-disruptive capability to download and selectively install LIC updates.

The z14 RAS features provide unique high-availability and non-disruptive operational capabilities that differentiate the Z servers in the marketplace. z14 RAS enhancements are made on many components of the CPC (processor chip, memory subsystem, I/O and service) in areas like error checking, error protection, failure handling, error checking, faster repair capabilities, sparing, and cooling.

The ability to cluster multiple systems in a Parallel Sysplex takes the commercial strengths of the IBM Z hardware and z/OS Operating System to higher levels of system management, scalable growth, and continuous availability.

## 2.4 z14 System Design

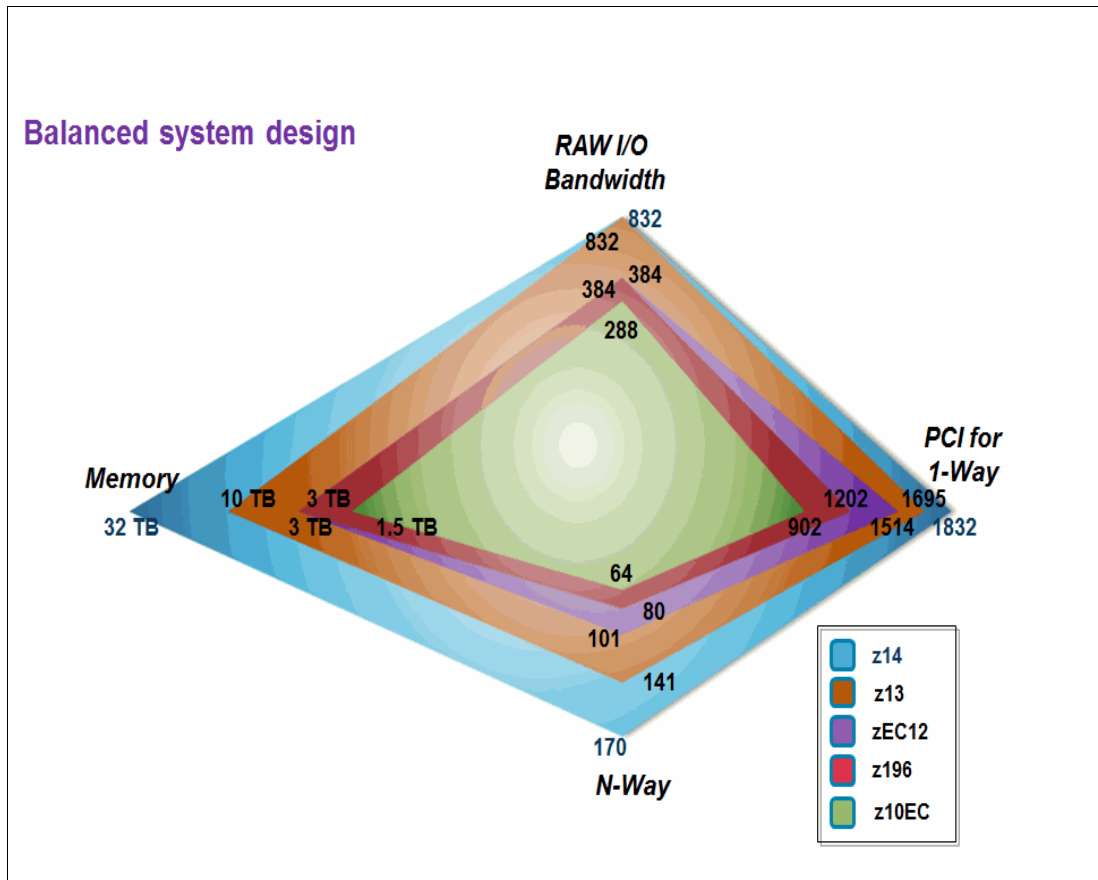


Figure 2-2 z14 System Design

### System design

Figure 2-2 shows a four-axis comparison between the most recent IBM Z product lines: z14, z13, z12, z196 and z10EC. As you can see, there is growth on three of the four axes, which results in a balanced system design:

- ▶ PCI (performance capacity index) per PU grew approximately 8%. Using Large Systems Performance Reference (LSPR) measurements, new measurements with z/OS V2R2, and re-evaluating the calculation for the mixed workload, the z14 uniprocessor PCI is 1832.
- ▶ The maximum number of processors grew from 141 to 170
- ▶ Central storage grew from 10 TB to 30 TB.
- ▶ The aggregate I/O remained constant as 832 GB/sec.

Central storage growth is proportionally larger than the other three resources. One of the reasons behind this increase is that in heavy commercial data processing environments, and with the trend in analytics to have large in-memory databases, central storage can really improve the overall performance by drastically decreasing the I/O rate. In other words, central storage is good for the overall performance health of the entire system.

## 2.5 Processor unit (PU) instances

As mentioned previously the z14 has up to 170 PUs that can be characterized. These PUs are physically identical. However, at POR, it is possible to have different types of PUs enabled through microcode or LIC. These instances are also called *specialty engines*. The major reason for such engines is to lower the overall total cost of computing on the mainframe platform. They can be ordered by customers and are known as *characterized*. Following are the PU instances that are possible to characterize:

- CPU** A general purpose processor that can execute all the possible z14 supported 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.
- ICF** This type of PU is only able to execute Coupling Facility Control Code (CFCC) operating system. The CFCC is loaded in a Coupling Facility LPAR from a copy in HSA; after this, the LPAR is activated.
- zIIP** This type of PU is only supported by z/OS. It is an offload engine designed to lower the cost of growing the platform. Various IBM and third-party workloads, including certain Db2 workloads, Java, MQ and numerous monitoring products can exploit these zIIPs.
- SAP** A System Assist Processor (SAP) is a PU that runs the Channel Subsystem Licensed Internal Code. A SAP manages the starting of I/O operations required by operating systems running in all logical partitions. It frees z/OS, for example (and the processor) from this role, and is “mainframe-unique”. z14 models have a variable number of standard SAPs configured.
- Spare** A spare PU is a PU that can replace, automatically and transparently, any failing PU in the same drawer, or in a different drawer. There are at least two spares per z14 server.
- 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 multi-drawer 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 or SAPs, on an existing drawer.
- ▶ You can add a new drawer concurrently (non-disruptively) and activate more CPs, IFLs, ICFs, zIIPs or SAPs.
- ▶ You can add a new drawer to provide additional memory and 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 these 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 processors without requiring the real specialty processor hardware.

## 2.6 z14 Hardware models

z14 Hardware models - Processor Unit Features							
Model	Drawers/ PUs	CPs/IFL S	zIIPs	ICFs	IFP	Standard SAPs	Standard Spares
M01	1 x 41	0-33	0-22	0-33	1	5	2
M02	2 x 41	0-69	0-46	0-69	1	10	2
M03	3 x 41	0-104	0-70	0-104	1	15	2
M04	4 x 41	0-140	0-94	0-140	1	20	2
M05	4 x 49	0-170	0-112	0-170	1	23	2

A minimum of one CP, ICF, or ICF must be purchased on every model.  
 Two zIIPs may be purchased for each CP purchased if PUs are available.  
 The IFP is conceptually an additional, special purpose SAP – used by some PCIe I/O features

Figure 2-3 z14 Hardware models

### z14 Hardware model

As shown in Figure 2-3, the z14 server hardware model nomenclature is based on the number of drawers installed in each server. These drawers have 41 Processor units (PUs), except for the M05. The PUs are called characterized and can take any PU personality. The z14 has five models with a total of 269 capacity settings available:

**Model M01:** one drawer with 41 PUs, 5 standard SAPs and 2 standard spares, 1 IFP, then 33 characterized PUs.

**Model M02:** two drawers with 41 PUs each, 10 standard SAPs and 2 standard spares, 1 IFP, then 69 characterized PUs.

**Model M03:** three drawers with 41 PUs each, 15 standard SAPs and 2 standard spares, 1 IFP, then 104 characterized PUs.

**Model M04:** four drawers with 41 PUs each, 20 standard SAPs and 2 standard spares, 1 IFP, then 140 characterized PUs.

**Model M05:** four drawers with 49 PUs each, 23 standard SAPs and 2 standard spares, 1 IFP, then 170 characterized PUs.

As mentioned, the z14 hardware model names indicate the number of drawers present (except for the M05 Special build system), and gives no indication of the actual number of active processors.

## 2.7 z14 Sub-capacity models

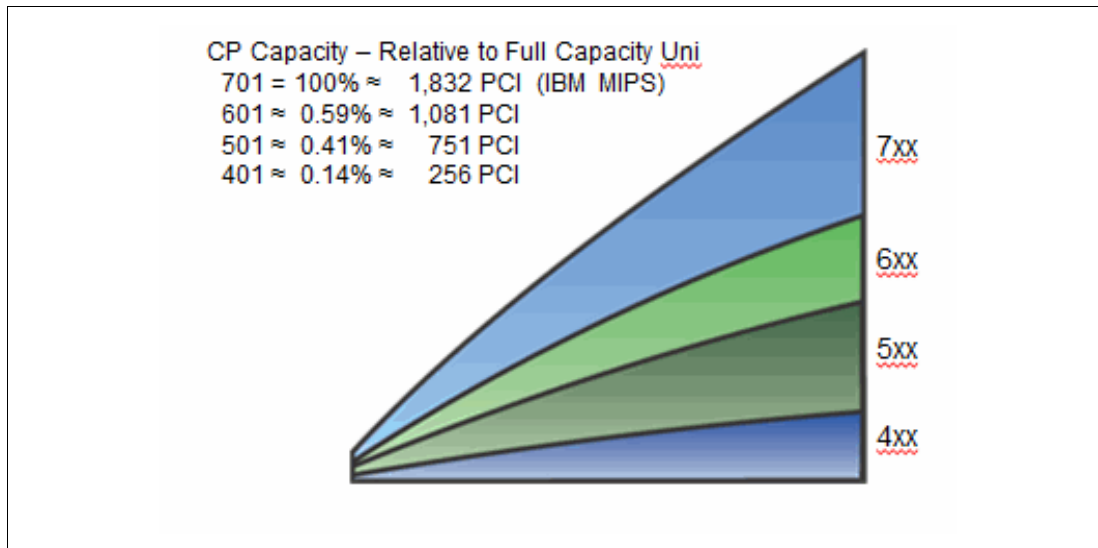


Figure 2-4 Sub-capacity models

### Sub-capacity models

Sub-capacity models are servers in which the instruction execution rate is controlled to increase “cycles per instruction”. It should be emphasized that the sub-capacity CPs are not actually slower - the clock frequency doesn’t change, but they have a lower capacity for performing work due to the change in execution rate.

On the z14, there are 170 (which is the number of available PUs) full capacity settings and there are an additional 99 sub-capacity settings (33 each in the 4xx, 5xx and 6xx range).

### Software model numbers

For software billing purposes only, there is a Model Capacity Indicator (MCI, also called a software model number) associated with the number of PUs that are characterized as processors.

There is no affinity between the hardware model and the number of activated CPs. For example, it is possible to have a two-drawer Model M02 (82 characterized PUs) but with only 13 processors, so for software billing purposes, the machine might report as a Model 713, 613, 513 or 413 depending on the capacity model chosen by the customer.

The software model number (Nxx) follows these rules:

- ▶ N = the capacity setting of the engine:
  - 7xx = 100%
  - 6xx ~ 59%
  - 5xx ~ 41%
  - 4xx ~ 14%
- ▶ xx = the number of PUs characterized as processors in the server

After xx exceeds 33, then all processor engines are full capacity.

## List of model capacity identifiers

The following Table lists all the models and capacity settings available and shows that regardless of the number of CPC drawers, a configuration with one characterized CP is possible. For example, model 7H0 might have only one PU characterized as a CP, or a two-book machine (M02) with 13 CPs enabled (out of a possible 82), could be a sub-capacity model 413, 513, or 613 or full-capacity model 713.

z14 Hardware Model	Model Capacity identifier
M01	701 - 733, 601 - 633, 501 - 533, and 401 - 433
M02	701 - 769, 601 - 633, 501 - 533, and 401 - 433
M03	701 - 7A5, 601 - 633, 501 - 533, and 401 - 433
M04	701 - 7E5, 601 - 633, 501 - 533, and 401 - 433
M05	701 - 7H0, 601 - 633, 501 - 533, and 401 - 433

**Note:** There is a model capacity identifier 400 which is used for ICF or IFL only models.

There are 270 (170 + 99 + 1 special case) possible combinations of capacity levels and numbers of processors. These offer considerable overlap in absolute capacity, providing different ways to achieve the required capacity. For example, a specific capacity (expressed as MSUs) might be obtained with a single faster CP or with three sub-capacity CPs. The single CP server 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 a mixed batch/online workload.

There are a number of conditions that apply to the sub-capacity models.

- All processors must be the same capacity setting size within one z14.
- They are only available for the models with fewer than 33 processors.
- Models with 34 CPs or greater must be full capacity.

## Model capacity identifier and MSU value

All model capacity identifiers have a related MSU value. The MSU values are used to determine the software license charge for MLC software. Tables with MSU values are available on the Mainframe Exhibits for IBM Servers website:

<http://www.ibm.com/systems/z/resources/swprice/reference/exhibits/hardware.html>

## 2.8 z14 frames, drawers, and I/O

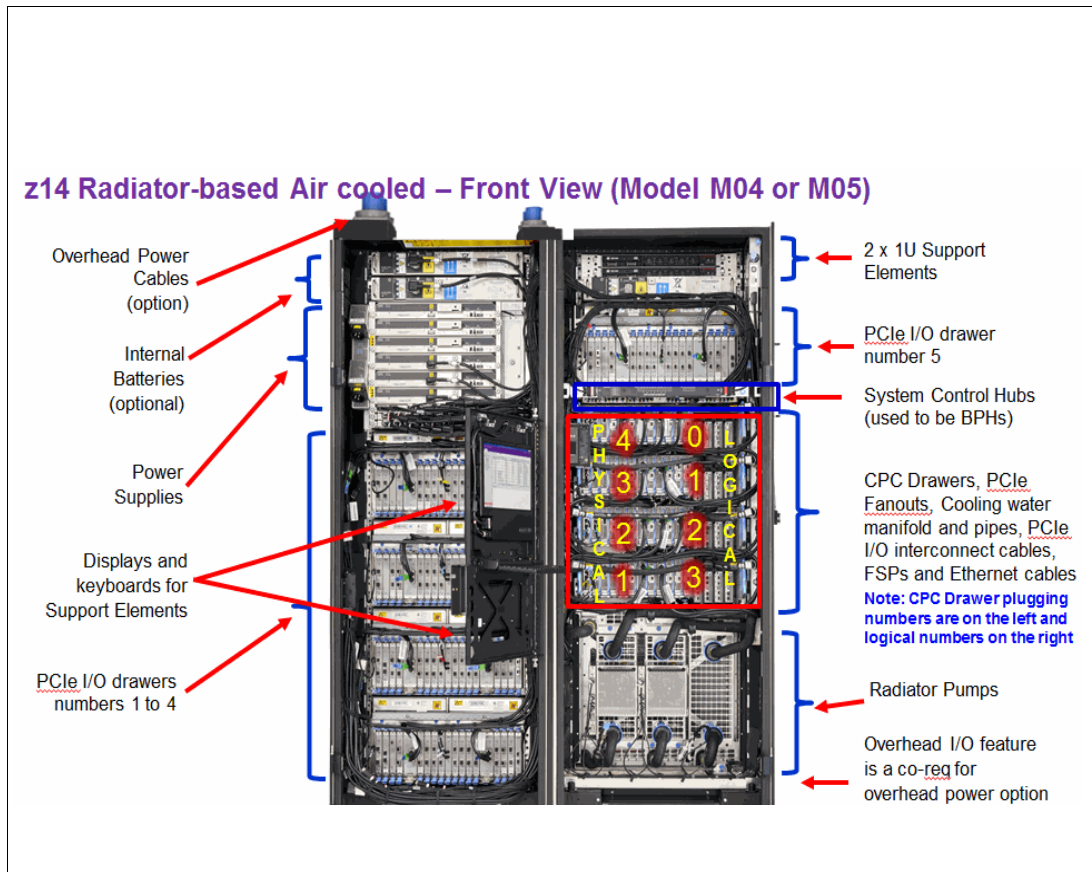


Figure 2-5 z14 frames

### z14 frames

The z14 is always a two-frame system. The *A Frame* and the *Z Frame*. It can be delivered as an air-cooled system or as a water-cooled system. The radiator-cooled z14 models support installation on raised floor and non-raised floor environments. For water-cooled models, installation is only available on a raised floor.

The A-Frame can house up to four Processor drawers, central storage, a PCIe I/O drawer (Peripheral Component Interconnect Express), cooling hardware and two 1U servers (Support Elements).

The Z-Frame can house up to four PCIe I/O drawers, the power supplies and the optional Internal Battery Facility (IBF).

The PCIe I/O drawers contain I/O cards with I/O processors, known as *channels*.

In addition, the z14 offers top-exit options for the I/O and power cables. These options give you more flexibility in planning where the system resides, potentially freeing you from running cables under a raised floor and increasing air flow over the system.

Figure 2-5 shows an internal, front view of the two frames of an air-cooled z14 system with the maximum five PCIe I/O drawers, including the top-exit power cable options.

### ***Optional Internal 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 actual amount of time is dependent on the number of I/O drawers. 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.

### **PCIe I/O drawers**

The z14 contains an I/O subsystem infrastructure that uses PCIe I/O drawers that support up to 32 PCIe I/O features each, and the ability to have up to 5 of these PCIe I/O drawers with the two frames of the z14, delivering support for a total of 160 I/O features.

PCIe I/O drawers provide increased I/O granularity and capacity flexibility and can be concurrently added and removed in the field.

## **2.9 Processor Drawers**

The z14 is a multiple processor (CPC) drawer system that can hold up to four drawers in the A Frame.

Each CPC drawer contains the following elements:

- ▶ Single chip modules:
  - Five or six PU single chip modules, each containing seven, eight, nine, or ten active processor unit cores.
  - One storage controller single chip module, with a total of 672 MB L4 cache.

Single chip modules are further described in the section 2.10, “Single Chip Module (SCM)” on page 91. 2

- ▶ Memory:
  - A minimum of 320GB and a maximum of 32 TB of memory (excluding 192GB HSA) is available for client use. See Table 2-1 on page 93 for details.
  - Either 15, 20, or 25 memory DIMMs are plugged in a CPC drawer.
- ▶ Fanouts:

Fanouts are interfaces to connect either the internal PCIe I/O drawers or externally to another IBM Z processors.

The CPC drawer provides up to 10 PCIe Gen3 fanout adapters to connect to the PCIe I/O drawers and ICA SR coupling links, and up to four InfiniBand fanout adapters for 12x InfiniBand and 1x InfiniBand coupling links.

Each fanout has one, two, or four ports:

- One-port PCIe 16 GBps I/O fanout, each supporting one domain in 32-slot PCIe I/O drawers (internal connection).
- ICA SR two-port PCIe fanout for coupling links (two links, 8 GBps each).



- HCA3-O 12x InfiniBand fanout for coupling links (two ports at 6 GBps each).
- HCA3-O LR 1x InfiniBand fanout for coupling links (four ports, 5 Gbps each).
- ▶ Two Distributed Converter Assemblies (DCA)

DCAs provide power to the CPC drawer. Loss of one DCA leaves enough power to satisfy the power requirements of the entire drawer. The DCAs can be concurrently maintained.

- ▶ Two Flexible Support Processors (FSP)

FSPs provide redundant interfaces to the internal management network.

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 is installed.

## 2.10 Single Chip Module (SCM)

A Processor drawer (CPC Drawer) has five PU single chip modules (SCMs) for models M01 through M04 or six PU SCMs for model M05, and one storage control (SC) SCM. Each PU SCM has seven, eight, nine, or ten active PU cores, and L1, L2, and L3 caches.

The SC SCM holds the L4 cache, and the number of active PU cores on each of the PU SCMs can also range from 7 to 10 for all models.

Each SC SCM includes 672 MB shared eDRAM cache. The SC SMC is configured to provide a single 672 MB L4 cache that is shared by all PU cores in the CPC drawer. This amount of cache provides a total of 2.68GB of cache if all four CPC drawers are implemented, yielding outstanding SMP scalability for real-world workloads.

## 2.11 Processor Unit (PU)

Processor unit (PU) is the generic term for an IBM z/Architecture processor. Each PU is a superscalar processor with the following attributes:

- ▶ Up to six instructions can be decoded per clock cycle.
- ▶ Up to ten instructions can be in execution per clock cycle.
- ▶ Instructions can be issued out-of-order. The PU uses a high-frequency, low-latency pipeline, providing robust performance across a wide range of workloads.
- ▶ Memory accesses might not be in the same instruction order (out-of-order operand fetching).
- ▶ Most instructions flow through a pipeline with varying numbers of steps for different types of instructions. Several instructions can be in execution at any moment, subject to the maximum number of decodes and completions per cycle.

### PU cache

The on-chip cache for the PU (core) works in this way:

- ▶ Each PU core has an L1 cache (private) that is divided into a 128 KB cache for instructions and a 128 KB cache for data.

- ▶ Each PU core has a private L2 cache, with 4 MB D-cache (“D” stands for data) and 2 MB I-cache (“I” stands for instruction).
- ▶ Each PU SCM contains a 128 MB L3 cache that is shared by all PU cores in the SCM. The shared L3 cache uses eDRAM.

This on-chip cache implementation optimizes system performance for high-frequency processors, with cache improvements, new Translation/TLB2 design, pipeline optimizations, and better branch prediction.

### **PU sparing**

Hardware fault detection is embedded throughout the design, and is combined with comprehensive instruction-level retry and dynamic PU sparing. This function provides the reliability and availability that is required for true mainframe integrity.

### **On-chip cryptographic hardware**

Dedicated on-chip cryptographic hardware for each PU core includes extended key and hash sizes for the Advanced Encryption Standard (AES) and Secure Hash Algorithm (SHA), and support for UTF8 to UTF16 conversion. This cryptographic hardware is available with any processor type, for example CP, IBM zIIP, and IFL.

### **Software support**

The z14 PUs provide full compatibility with existing software for z/Architecture, and extend the Instruction Set Architecture (ISA) to enable enhanced functionality and performance. Several hardware instructions that support more efficient code generation and execution are introduced in the z14.

### **PU characterization**

PUs are ordered in single increments. The internal system functions, which are based on the configuration that is ordered, characterize each PU into one of various types during system initialization, which is often called a power-on reset (POR) operation. Characterizing PUs dynamically without a POR is possible by using a process called *Dynamic Processor Unit Reassignment*. A PU that is not characterized cannot be used. As discussed previously, each PU can be characterized as follows:

- ▶ Central processor (CP)
- ▶ Integrated Facility for Linux (IFL) processor
- ▶ IBM Z Integrated Information Processor (zIIP)
- ▶ Internal Coupling Facility (ICF)
- ▶ System assist processor (SAP)
- ▶ Integrated firmware processor (IFP)

## **2.12 Memory**

Maximum physical memory size is directly related to the number of CPC drawers in the system. Typically, a system has more memory installed than was ordered because part of the installed memory is used to implement the redundant array of independent memory (RAIM) design. On the z14, this configuration results in up to 8 TB of available memory per CPC drawer and up to 32 TB for a four-drawer system.

The hardware system area (HSA) on the z14 has a fixed amount of memory (192 GB) that is managed separately from client memory. However, the maximum amount of orderable memory can vary from the theoretical number due to dependencies on the memory granularity.

Table 2-1 lists the maximum memory sizes for each z14 model.

*Table 2-1 Memory Size*

<b>Model</b>	<b>CPC Drawers</b>	<b>Maximum Memory</b>
M01	1	8 TB
M02	2	16 TB
M03	3	24 TB
M04	4	32 TB
M05	4	32 TB

On z14 systems, the granularity for memory increases is 64 - 512 GB. Table 2-2 shows the memory increments, depending on installed memory.

*Table 2-2 Memory increments and range*

<b>Memory Increment (GB)</b>	<b>Memory Range (GB)</b>
64	256-384
128	384-896
256	896-2944
512	2944-32576

## 2.13 Power and Cooling

The z14 power and cooling system is a continuation of IBM z13 with the addition of improvements in availability, temperature tolerance, and vectored (directional) air output. The air-cooled z14 server now has a radiator unit (RU) N+1 design for the pumps and blowers. With the new rear cover design for vectored air output, you can choose whether the output air goes up or down. The water-cooled system is still an option for the z14 server. The Top Exit Power feature is available for the z14 server. Combined with the Top Exit I/O Cabling feature, it gives you more options when you are planning your computer room cabling.

### 2.13.1 Power

The system operates with two fully redundant power supplies. One is in the front side of the Z frame, and the other is in the rear side of the Z frame. Each power supply has either one or two power cords. The number of power cords that are required depends on the system configuration. The total loss of one power supply has no impact on system operation.

Systems that specify two power cords can be brought up with one power cord and continue to run. The larger systems that have a minimum of four BPR (Bulk Power Regulators) pairs that are installed must have four power cords installed. Systems that specify four power cords can

be started with two power cords on the *same side* with sufficient power to keep the system running.

Power cords attach to either a three-phase, 50/60 Hz, 200 - 480 V AC power source, or a 380 - 520 V DC power source.

A Balanced Power Plan Ahead feature is available for future growth, helping to ensure adequate and balanced power for all possible configurations. With this feature, system downtime for upgrading a server is eliminated by including the maximum power requirements in terms of BPRs and power cords to your installation.

For ancillary equipment, such as the Hardware Management Console, its display, and its switch, extra single-phase outlets are required.

The power requirements depend on the cooling facility that is installed, and on the number of CPC drawers and I/O units that are installed. For more information on power requirements, see Installation Manual for Physical Planning, GC28-6965-00a, available on IBM Resource Link®.

### Power estimation tool

The power estimation tool for the z14 server allows you to enter your precise server configuration to obtain an *estimate* of power consumption. Log in to IBM Resource link with your user ID. Click **Planning** -> **Tools** -> **Power Estimation Tools**. Specify the quantity for the features that are installed in your system. This tool estimates the power consumption for the specified configuration. The tool does not verify that the specified configuration can be physically built.

**Note:** The exact power consumption for your system varies. The object of the tool is to estimate the power requirements to aid you in planning for your system installation. After your system is installed, then the actual power consumption can be confirmed by using the HMC Monitor Dashboard task.

## 2.13.2 Cooling

As discussed previously, the z14 is available as either air cooled or water cooled, depending on the customers' requirements. The selection of air-cooled models or water-cooled models is done when ordering, and the appropriate equipment is factory-installed. An MES (conversion) from an air-cooled model to a water-cooled model and vice-versa is not allowed.

In all z14 servers, the CPC drawer, SC SCMs, PCIe I/O drawers, I/O drawers, and power enclosures are all cooled by forced air with blowers that are controlled by the Move Device Assembly (MDA).

The PU SCMs in the CPC drawers are cooled by water. The internal closed water loop takes heat away from PU SCMs by circulating water between the radiator heat exchanger and the cold plate that is mounted on the PU SCMs.

### Air-cooled models

Although the PU SCMs are cooled by an internal closed loop water system, the heat is exhausted into the room from the radiator heat exchanger by forced air with blowers.

## Water-cooled models

z14 servers are available as water-cooled systems. With WCU technology, z14 servers can transfer most of the heat that they generate into the building's chilled water, which effectively reduces the heat output to the computer room.

Unlike the radiator in air-cooled models, an IBM z14 has two water loops: An internal closed water loop (same as with the air-cooled models) and an external (chilled) water loop. The external water loop connects to the client-supplied building's chilled water. The internal water loop circulates between the water cooling unit (WCU) heat exchanger and the PU SCMs cold plates. The loop takes heat away from the PU SCMs and transfers it to the external water loop in the WCU's heat exchanger.

In addition to the PU SCMs, the internal water loop also circulates through two heat exchangers that are in the path of the exhaust air in the rear of the frames. These heat exchangers remove approximately 60% - 65% of the residual heat from the I/O drawers, PCIe I/O drawers, the air-cooled logic in the CPC drawers, and the power enclosures. Almost two-thirds of the total heat that is generated can be removed from the room by the chilled water.

## 2.14 Upgrades

Capacity upgrades are possible on all IBM z14 systems and may be required for various reasons. For example, a business might grow steadily, year after year, or activity spikes can happen during marketing campaigns.

Upgrades can be categorized as described in the following section.

### Permanent and temporary upgrades

Permanent and temporary upgrades are different types of upgrades that can be used in different situations. For example, a growing workload might require more memory, more I/O cards, or more processor capacity. However, only a short-term upgrade might be necessary to handle a peak workload, or to temporarily replace a system that is down during a disaster or data center maintenance. z14 servers offer the following solutions for such situations:

► Permanent:

- Miscellaneous equipment specification (MES):

The MES upgrade order is always performed by IBM personnel. The result can be either real hardware or installation of Licensed Internal Code Configuration Control (LICCC) to the system. In both cases, installation is performed by IBM personnel.

- Customer Initiated Upgrade (CIU):

Using the CIU facility for a system requires that the online Capacity on Demand (CoD) buying feature is installed on the system (this is usually done at the time of ordering). The CIU facility supports only LICCC upgrades.

► Temporary:

All temporary upgrades are LICCC-based. There are three types of temporary upgrade available. See 2.14.2, "Temporary upgrades" on page 97.

### Concurrent and non-disruptive upgrades

Depending on the impact on the system and application availability, upgrades can be classified in the following manner:

- ▶ Concurrent

In general, *concurrency* addresses the continuity of operations of the hardware part of an upgrade. For example, whether a system (hardware) is required to be turned off during the upgrade.

- ▶ Non-concurrent

This type of upgrade requires turning off the hardware that is being upgraded. Examples include model upgrades from any z14 model to the z14 M05 model, and certain physical memory capacity upgrades.

- ▶ Disruptive

An upgrade is considered *disruptive* when resources modified or added to an operating system image require that the operating system be restarted to configure the newly added resources.

- ▶ Non-disruptive

*Non-disruptive* upgrades do not require the software or operating system to be restarted for the upgrade to take effect. Therefore, even concurrent upgrades can be disruptive to operating systems or programs that do not support the upgrades while being non-disruptive to others.

## 2.14.1 Permanent upgrades

Permanent upgrades can be obtained by either ordering through an IBM representative or initiated by the client with CIU on IBM Resource Link.

Permanent upgrades that are ordered through an IBM representative are available for the following tasks:

- ▶ Add processor drawers
- ▶ Add Peripheral Component Interconnect Express (PCIe) drawers and features
- ▶ Add model capacity
- ▶ Add specialty engines
- ▶ Add memory
- ▶ Activate unassigned model capacity or IFLs
- ▶ Deactivate activated model capacity or IFLs
- ▶ Activate channels
- ▶ Activate cryptographic engines
- ▶ Change specialty engine (re-characterization)

Permanent upgrades initiated by the client through CIU allow you to add capacity to fit within your existing hardware.

- ▶ Add model capacity
- ▶ Add specialty engines
- ▶ Add memory
- ▶ Activate unassigned model capacity or IFLs
- ▶ Deactivate activated model capacity or IFLs

## 2.14.2 Temporary upgrades

z14 servers offer three types of temporary upgrades:

- ▶ On/Off Capacity on Demand (OOCOD):

This offering allows you to temporarily add more capacity or specialty engines to cover seasonal activities, period-end requirements, peaks in workload, or application testing. This temporary upgrade can be ordered only by using the CIU application through Resource Link.

- ▶ Capacity Backup (CBU):

This offering allows you to replace model capacity or specialty engines in a backup system that is used in an unforeseen loss of system capacity because of a disaster.

- ▶ Capacity for Planned Event (CPE):

This offering allows you to replace model capacity or specialty engines because of a relocation of workload during system migrations or a data center move.

CBU or CPE temporary upgrades can be ordered by using the CIU application through Resource Link or by calling your IBM marketing representative.

Temporary upgrade capacity changes can be billable or a replacement.

### Billable capacity

To handle a peak workload, you can activate up to double the purchased capacity of any processor unit (PU) type temporarily. You are charged daily.

The one billable capacity offering is On/Off Capacity on Demand (OOCOD).

### Replacement capacity

When a processing capacity is lost in another part of an enterprise, replacement capacity can be activated. It allows you to activate any PU type up to your authorized limit.

The following offerings are the two replacement capacity offerings:

- ▶ Capacity Backup
- ▶ Capacity for Planned Event

## 2.14.3 Concurrent upgrades

Concurrent upgrades on z14 servers can provide more capacity with no system outage. In most cases, with prior planning and operating system support, a concurrent upgrade can be non-disruptive to the operating system.

The concurrent capacity growth capabilities that are provided by z14 servers include, but are not limited to, these benefits:

- ▶ Enabling the meeting of new business opportunities
- ▶ Supporting the growth of smart and cloud environments
- ▶ Managing the risk of volatile, high-growth, and high-volume applications
- ▶ Supporting 24 x 7 application availability
- ▶ Enabling capacity growth during *lockdown* or *frozen* periods
- ▶ Enabling planned-downtime changes without affecting availability

This capability is based on the flexibility of the design and structure, which allows concurrent hardware installation and Licensed Internal Code (LIC) control over the configuration.

The sub-capacity models allow more configuration granularity within the family. The added granularity is available for models that are configured with up to 33 CPs, and provides 99 extra capacity settings. Sub-capacity models provide for CP capacity increase in two dimensions that can be used together to deliver configuration granularity. The first dimension is adding CPs to the configuration. The second is changing the capacity setting of the CPs currently installed to a higher model capacity identifier.

z14 servers allow the concurrent and non-disruptive addition of processors to a running logical partition (LPAR). As a result, you can have a flexible infrastructure in which you can add capacity without pre-planning. This function is supported by z/OS, z/VM, and z/VSE. There are two ways to accomplish this addition:

- ▶ With planning ahead for the future need of extra processors. In the LPAR's profile, reserved processors can be specified. When the extra processors are installed, the number of active processors for that LPAR can be increased without the need for a partition reactivation and initial program load (IPL).
- ▶ Another (easier) way is to enable the dynamic addition of processors through the z/OS LOADxx member. Set the **DYNCPADD** parameter in member LOADxx to ENABLE. z14 servers support dynamic processor addition in the same way that the z13, zEC12, z196, and z10 support it.

#### 2.14.4 Customer Initiated Upgrade facility

The CIU facility is an IBM online system through which you can order, download, and install permanent and temporary upgrades for IBM Z servers. Access to and use of the CIU facility requires a contract between the client and IBM. The use of the CIU facility for a system requires that the online CoD buying feature code is installed on the system. Although it can be installed on your z14 servers at any time, it is generally added when ordering a z14 server.

After you place an order through the CIU facility, you receive a notice that the order is ready for download. You can then download and apply the upgrade by using functions that are available through the Hardware Management Console (HMC), along with the RSF. After all the prerequisites are met, the entire process, from ordering to activation of the upgrade, is performed by the client.

After download, the actual upgrade process is fully automated and does not require any onsite presence of IBM SSRs.

The CIU facility supports LICCC upgrades only. It does not support I/O upgrades. All additional capacity that is required for an upgrade must be previously installed.

##### **Permanent upgrades**

Permanent upgrades can be ordered by using the CIU facility. Through the CIU facility, you can generate online permanent upgrade orders to concurrently add processors (CPs, ICFs, zIIPs, IFLs, and SAPs) and memory, or change the model capacity identifier. You can do so up to the limits of the installed processor drawers on an existing system.

##### **Temporary upgrades**

The base model z14 server describes permanent and dormant capacity using the capacity marker and the number of PU features installed on the system. Up to eight temporary offerings can be present. Each offering has its own policies and controls, and each can be activated or deactivated independently in any sequence and combination. Although multiple



offerings can be active at any time, if enough resources are available to fulfill the offering specifications, only one OOCOD offering can be active at any time.

### 2.14.5 Capacity Backup

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 z14 server.

CBU is a quick, temporary activation of processor units (CP, zIIP, IFL, ICF, SAP) in the face of a loss of processing capacity due to an emergency or disaster/recovery situation. When a z14 has the CBU feature enabled, the client is entitled to a number of tests over a period of time to validate that the CBU works as expected.

**Note:** CBU is for disaster/recovery purposes only, and cannot be used for peak load management of customer workload under the terms and conditions which the client has signed.

### 2.14.6 Capacity for Planned Events (CPE)

A CPE plan is offered with the z14 to provide replacement backup capacity for planned downtime events and allows for replacement capacity to be installed temporarily on another z14 in the customer's environment.

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, 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 z14 non-disruptively. 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.





## IBM Z connectivity

This chapter discusses the connectivity options available to connect the IBM z14 with I/O control units, another IBM Z and the network (LAN and WAN). It also briefly highlights the hardware and software components involved in such connectivity.

The IBM Z family includes the new IBM z14 (z14) as well as the IBM z13s (z13s), IBM z13 (z13), zEnterprise zEC12, zBC12, z196, z114.

Unless otherwise stated, the term *OSA* applies to all OSA-Express features throughout this book.

For more detail on IBM Z connectivity, see the *IBM Z Connectivity Handbook*, SG24-5444.

### 3.1 I/O channel overview

One of the many key strengths of the IBM Z platform is the ability to deal with large volumes of simultaneous I/O operations. The channel subsystem (CSS) provides the function for Z platforms to communicate with external I/O and network devices and manage the flow of data between those external devices and system memory. This goal is achieved by using a system assist processor (SAP) that connects the CSS to the external devices.

The SAP uses the I/O configuration definitions loaded in the hardware system area (HSA) of the system to identify the external devices and the protocol they support. The SAP also monitors the queue of I/O operations passed to the CSS by the operating system.

Using a SAP, the processing units (PUs) are relieved of the task of communicating directly with the devices, so data processing can proceed concurrently with I/O processing.

Increased system performance demands higher I/O and network bandwidth, speed, and flexibility, so the CSS evolved along with the advances in scalability of the Z platforms. The z/Architecture provides functions for scalability in the form of multiple CSS that can be configured within the same IBM Z platform,

### 3.1.1 I/O hardware infrastructure

I/O hardware connectivity is implemented through several I/O features, some of which support the Peripheral Component Interconnect Express (PCIe) Gen 3 standards. They are housed in PCIe I/O drawers.

PCIe I/O drawers allow a higher number of features (four times more than the I/O drawer and a 14% increase over the I/O cage on previous IBM z13) and increased port granularity. Each drawer can accommodate up to 32 features in any combination. They are organized in four hardware domains per drawer, with eight features per domain. The PCIe I/O drawer is attached to a PCIe fanout in the CPC drawer, with an interconnection speed of 8 GBps with PCIe Gen2 and 16 GBps with PCIe Gen3. PCIe I/O drawers can be installed and repaired concurrently in the field.

## 3.2 I/O connectivity features

The most common attachment to an Z I/O channel is a storage control unit (CU), which can be accessed through a Fibre Connection (IBM FICON) channel, for example. The CU controls I/O devices such as disk and tape drives.

IBM zHyperLink is a new technology that provides a low-latency point-to-point connection from a z14 to an IBM DS8880 Disk Controller. It provides a 5-fold reduction in I/O service time for *read* requests.

System-to-system communications are typically implemented by using the IBM Integrated Coupling Adapter (ICA SR), Coupling Express Long Reach (CE LR), InfiniBand (IFB) coupling links, Shared Memory Communications, and FICON channel-to-channel (FCTC) connections.

The Internal Coupling (IC) channel, IBM HiperSockets, and Shared Memory Communications can be used for communications between logical partitions within the Z platform.

The Open Systems Adapter (OSA) features provide direct, industry-standard Ethernet connectivity and communications in a networking infrastructure.

The 10GbE RoCE Express2 and 10GbE RoCE Express features provide high-speed, low-latency networking fabric for IBM z/OS-to-z/OS shared memory communications.

Many features have an integrated processor that handles the adaptation layer functions required to present the necessary features to the rest of the system in a uniform manner. Therefore, all the operating systems have the same interface with the I/O subsystem.

The z14, z13, z13s, zEC12, and zBC12 support industry-standard PCIe adapters called *native PCIe adapters*. For native PCIe adapter features, there is no adaptation layer, but the device driver is present in the operating system. The adapter management functions (such as diagnostics and firmware updates) are provided by Resource Groups.

There are four Resource Groups on z14, and there are two Resource Groups on z13, z13s, zEC12, and zBC12. The Resource Groups are managed by an integrated firmware processor that is part of the system's base configuration.

The following sections briefly describe connectivity options for the I/O features available on the z14 platforms.

### 3.2.1 FICON EXPRESS

The Fibre Channel connection (FICON) Express features were originally designed to provide access to the traditional mainframe extended count key data (IBM ECKD) devices, and FICON channel-to-channel (CTC) connectivity. Then came support for access to Small Computer System Interface (SCSI) devices. This innovation was followed by support for High-Performance FICON for IBM Z (zHPF) for OLTP I/O workloads that transfer small blocks of fixed-size data. IBM Z platforms have continued to build on this I/O architecture with each new generation of IBM Z.

The FICON implementation enables full-duplex data transfer. In addition, multiple concurrent I/O operations can occur on a single FICON channel. FICON link distances can be extended by using various solutions.

The FICON features on IBM Z also support full fabric connectivity for the attachment of SCSI devices by using the Fibre Channel Protocol (FCP). Software support is provided by IBM z/VM, IBM z/VSE, and Linux on IBM Z operating systems.

### 3.2.2 zHyperLink Express

IBM zHyperLink is a technology that provides a low-latency point-to-point connection from a z14 to an IBM DS8880. The transport protocol is defined for reading and writing ECKD data records. It provides a 5-fold reduction in I/O services time for *read* requests.

The zHyperLink Express feature is a native PCIe adapter and can be shared by multiple LPARs.

On the z14, the zHyperLink Express feature is installed in the PCIe I/O drawer, on the IBM DS8880, the fiber optic cable connects to a zHyperLink PCIe interface in I/O bay.

The zHyperLink Express feature has the same qualities of service as do all Z I/O channel features. zHyperLink Express is not a replacement for FICON and in fact FICON attachments to the same DS8880 are a requirement to exploit this feature.

### 3.2.3 Open Systems Adapter-Express

The Open Systems Adapter-Express (OSA-Express) features are the I/O feature through which data is exchanged between the Z platforms and devices in the network. OSA-Express5S and OSA-Express6S are the two features supported on the IBM z14 which provide direct, industry-standard LAN connectivity in a networking infrastructure.

The OSA-Express features bring the strengths of the Z family, such as security, availability, and enterprise-wide access to data to the LAN environment. OSA-Express provides connectivity for the following LAN types:

- ▶ 1000BASE-T Ethernet (10/100/1000 Mbps)
- ▶ 1 Gbps Ethernet
- ▶ 10 Gbps Ethernet

### 3.2.4 HiperSockets

IBM HiperSockets technology provides seamless network connectivity to consolidate servers in an advanced infrastructure intra-server network. HiperSockets creates multiple independent, integrated, virtual LANs within a single IBM Z platform.

This technology provides high-speed connectivity between combinations of logical partitions or virtual servers. It eliminates the need for any physical cabling or external networking connection between these virtual servers. This *network within the box* concept minimizes network latency and maximizes bandwidth capabilities between z/VM, Linux on IBM Z, IBM z/VSE, and IBM z/OS images, or any combination of these.

HiperSockets use is also possible under the IBM z/VM operating system, which enables establishing internal networks between guest operating systems, such as multiple Linux servers.

The z/VM virtual switch can transparently bridge a guest virtual machine network connection on a HiperSockets LAN segment. This bridge allows a single HiperSockets guest virtual machine network connection to directly communicate with other guest virtual machines on the virtual switch and with external network hosts through the virtual switch OSA port.

HiperSockets technology is implemented by IBM Z LIC, with the communication path in system memory and the transfer information between the virtual servers at memory speed.

### 3.3 Coupling links

Parallel Sysplex is a synergy between hardware and software - a highly advanced technology for clustering designed to enable the aggregate capacity of multiple z/OS systems to be applied against common workloads.

It consists of the following components:

- ▶ Parallel Sysplex-capable servers
- ▶ Coupling facilities (CF)
- ▶ Coupling links (to connect the servers to the CF)
- ▶ Server Time Protocol (STP) - a solution for a common time source
- ▶ Shared direct access storage device (DASD)
- ▶ Software, both system and subsystem

Parallel Sysplex cluster technology is a highly advanced, clustered, commercial processing system. It supports high-performance, multisystem, read/write data sharing, which enables the aggregate capacity of multiple z/OS systems to be applied against common workloads.

The systems in a Parallel Sysplex configuration are linked and can fully share devices and run the same applications. This feature enables you to harness the power of multiple IBM Z platforms as though they are a single logical computing system.

The architecture is centered around the implementation of a coupling facility (CF) that runs the coupling facility control code (CFCC) and high-speed coupling connections for intersystem and intra-system communications. The CF provides high-speed data sharing with data integrity across multiple IBM Z platforms.

Coupling Links connectivity for Parallel Sysplex on z14 use Coupling Express Long Reach (CE LR), Integrated Coupling Adapter Short Reach (ICA SR), and InfiniBand (IFB) technology.

### 3.3.1 Coupling Express Long Reach (CE LR)

The CE LR, first introduced on the z14 platform but also made available on IBM z13 and IBM z13s, is a PCIe native adapter that is used for long-distance coupling connectivity. CE LR uses a new coupling channel type: CL5. The CE LR feature uses PCIe Gen3 technology and is hosted in a PCIe I/O drawer.

The feature supports communication at unrepeated distances up to 10 km (6.2 miles) using 9  $\mu\text{m}$  single-mode fiber optic cables and repeated distances up to 100 km (62 miles) using IBM Z qualified DWDM vendor equipment. The coupling links can be defined as shared between images within a CSS or spanned across multiple CSSs in IBM Z.

### 3.3.2 Integrated Coupling Adapter: Short Reach (ICA SR)

The ICA SR, first introduced on the z13 platform, is used for short distance coupling connectivity and uses the coupling channel type CS5. The ICA uses PCIe Gen3 technology and is hosted in a PCIe I/O drawer.

The ICA SR supports a cable length of up to 150 meters and supports a link data rate of 8 GBps. The coupling links can be defined as shared between images within a CSS, or spanned across multiple CSS in IBM Z.

### 3.3.3 Host Channel Adapter3: Optical Long Reach (HCA3-O LR)

The HCA3-O LR is used for long distance coupling connectivity and uses the coupling channel type CIB. This feature is a fanout card (not the newer PCIe I/O) and is based on InfiniBand protocol.

This feature supports communication at unrepeated distances up to 10 km (6.2 miles) using 9  $\mu\text{m}$  single mode fiber optic cables and repeated distances up to 100 km (62 miles) using IBM Z qualified DWDM equipment. The coupling links can be defined as shared between images within a CSS, or spanned across multiple CSSs in IBM Z.

### 3.3.4 Host Channel Adapter3: Optical (HCA3-O)

The HCA3-O is used for short distance coupling connectivity and uses the coupling channel type CIB. This feature is a fanout card (not the newer PCIe I/O) and is based on InfiniBand protocol.

This feature supports a cable length of up to 150 meters using industry standard OM3 50  $\mu\text{m}$  fiber optic cables. The coupling links can be defined as shared between images within a CSS, or spanned across multiple CSSs in IBM Z.

### 3.3.5 Internal coupling

Internal coupling links are used for internal communication between LPARs on the same system running coupling facilities (CF) and z/OS images. These links use the coupling channel type ICP. The connection is emulated in Licensed Internal Code (LIC) and provides for fast and secure memory-to-memory communications between LPARs within a single system. No physical cabling is required.

## 3.4 Shared Memory Communications

Shared Memory Communications (SMC) on IBM Z platforms is a technology that can improve throughput by accessing data faster with less latency. SMC reduces CPU resource consumption compared to traditional TCP/IP communications. Furthermore, applications do not need to be modified to gain the performance benefits of SMC.

SMC allows two peers to send and receive data by using system memory buffers that each peer allocates for its partner's use. Two types of SMC protocols are available on the IBM Z platform, both of which use shared memory architectural concepts, eliminating TCP/IP processing in the data path, yet preserving TCP/IP quality of service (QoS) for connection management purposes.

### 3.4.1 SMC-Remote Direct Memory Access (SMC-R)

SMC-R is a protocol for Remote Direct Memory Access (RDMA) communication between TCP socket endpoints in LPARs in different systems. SMC-R runs over networks that support RDMA over Converged Ethernet (RoCE). It allows existing TCP applications to benefit from RDMA without requiring modifications. SMC-R provides dynamic discovery of the RDMA capabilities of TCP peers and automatic setup of RDMA connections that those peers can use.

The 10GbE RoCE Express2 and 10GbE RoCE Express features provide the RoCE support needed for LPAR-to-LPAR communication across Z platforms.

### 3.4.2 SMC-Direct Memory Access (SMC-D)

SMC-D implements the same SMC protocol that is used with SMC-R to provide highly optimized intra-system communications. Where SMC-R uses RoCE for communicating between TCP socket endpoints in separate systems, SMC-D uses Internal Shared Memory (ISM) technology for communicating between TCP socket endpoints in the same system.

ISM provides adapter virtualization to facilitate the intra-system communications. Hence, SMC-D does not require any additional physical hardware (no adapters, switches, fabric management, or PCIe infrastructure). Therefore, significant cost savings can be achieved when using the ISM for LPAR-to-LPAR communication within the same Z platform.

Both SMC protocols use shared memory architectural concepts, eliminating TCP/IP processing in the data path, yet preserving TCP/IP quality of service (QoS) for connection management purposes.

## 3.5 Special-purpose feature support

In addition to the I/O connectivity features, several special purpose features are available that can be installed in the PCIe I/O drawers:

- ▶ Crypto Express
- ▶ Flash Express
- ▶ zEDC Express



### 3.5.1 Crypto Express features

Integrated cryptographic features provide leading cryptographic performance and functions. Reliability, availability, and serviceability support is unmatched in the industry, and the cryptographic solution received the highest standardized security certification (FIPS 140-2 Level 4).

Crypto Express6S and Crypto Express5S (and previous generations) are tamper-sensing and tamper-responding programmable cryptographic features that provide a secure cryptographic environment. Each adapter contains a tamper-resistant hardware security module (HSM).

The HSM can be configured as a secure IBM Common Cryptographic Architecture (CCA) coprocessor, as a secure IBM Enterprise PKCS #11 (EP11) coprocessor, or as an accelerator.

Each Crypto Express6S and Crypto Express5S feature occupies one I/O slot in the PCIe I/O drawer. Crypto Express6S is supported on the z14. Crypto Express5S is supported on the z13 and z13s platforms.

### 3.5.2 Flash Express feature

Flash Express is an innovative optional feature that was introduced with the zEC12 and is also available on zBC12, z13, and z13s platforms. It improves performance and availability for critical business workloads that cannot afford any reduction in service levels. Flash Express is easy to configure, requires no special skills, and produces rapid time to value.

Flash Express implements storage-class memory (SCM) through an internal NAND Flash solid-state drive (SSD) in a PCIe adapter form factor. Each Flash Express feature is installed exclusively in the PCIe I/O drawer and occupies one I/O slot.

For availability reasons, the Flash Express feature must be ordered in pairs. A *feature pair* provides 1.4 TB of usable storage.

Flash Express storage is allocated to each partition in a manner similar to main memory allocation. The allocation is specified at the Hardware Management Console (HMC). z/OS can use the Flash Express feature as SCM for paging store to minimize the supervisor call (SVC) memory dump duration.

The z/OS paging subsystem supports a mix of Flash Express and external disk storage. Flash Express can also be used by Coupling Facility images to provide extra capacity for particular structures (for example, IBM MQ shared queues application structures).

Starting with the release of the z14, IBM has replaced Flash Express with Virtual Flash Memory (VFM). VFM utilizes HSA type memory instead of a pair of flash cards, and supports the same use cases as above.

### 3.5.3 zEDC Express feature

zEDC Express is an optional feature that is exclusive to the z14, z13, z13s, zEC12, and zBC12 systems. It provides hardware-based acceleration of data compression and decompression. That capability improves cross-platform data exchange, reduces processor use, and saves disk space.

A minimum of one feature can be ordered, and a maximum of 16 can be installed on the system on the PCIe I/O drawer. A zEDC Express feature can be shared by up to 15 LPARs. z/OS version 2.1 and later support the zEDC Express feature. z/VM 6.3 with PTFs and later also provides guest exploitation.

## 3.6 Channel Subsystem

The role of the channel subsystem (CSS) is to control communication of internal and external channels to control units and devices. The CSS configuration defines the operating environment for the correct execution of all system I/O operations.

The CSS provides the server communications to external devices through channel connections. The channels transfer data between main storage and I/O devices or other servers under the control of a channel program. The CSS allows channel I/O operations to continue independently of other operations within the central processors (CPs) and Integrated Facility for Linux processors (IFLs).

An important piece of the I/O z/Architecture is the SAP (System Assist Processor), which is a PU in charge of guaranteeing the start and end of an I/O operation. It is a unique mainframe feature that frees many processor cycles during I/O operations.

### 3.6.1 Multiple channel subsystems concept

The design of IBM Z offers considerable processing power, memory size, and I/O connectivity. In support of the larger I/O capability, the CSS concept is scaled up correspondingly. The increased size provides relief for the number of supported logical partitions (LPARs), channels, and devices available to the system.

A single channel subsystem allows the definition of up to 256 channel paths. To overcome this limit, the multiple channel subsystems concept was introduced. The z14 architecture provides for six channel subsystems. The structure of the multiple LCSSs provides channel connectivity to the defined LPARs in a manner that is transparent to subsystems and application programs. This configuration enables the definition of a balanced configuration for the processor and I/O capabilities.

The LCSS can have from 1 to 256 channels, and can be configured to 1 to 15 LPARs, except for CSS 5 which can only be configured with 10 LPARs.

Therefore, the six channel subsystems support a maximum of 85 LPARs. CSSs are numbered from 0 to 5, which is referred to as the *CSS image ID* (CSSID 0, 1, 2, 3, 4, or 5). These CSS are also referred to as *logical channel subsystems* (LCSS).

#### CSS elements

The CSS is composed of the following elements:

- ▶ Subchannels
- ▶ Channel paths
- ▶ Channel path identifier
- ▶ Control units
- ▶ I/O devices

## **Subchannels**

A *subchannel* provides the logical representation of a device to a program. It contains the information that is required for sustaining a single I/O operation. A subchannel is assigned for each device that is defined to the LPAR.

Multiple subchannel sets, described in “Multiple subchannel sets” on page 109, are available to increase addressability. Four subchannel sets per CSS are supported on a z14. Subchannel set 0 can have up to 65280 subchannels, and subchannel sets 1, 2, and 3 can have up to 65535 subchannels each.

## **Channel paths**

Each CSS can have up to 256 channel paths. A *channel path* is a single interface between a server and one or more control units. Commands and data are sent across a channel path to run I/O requests.

## **Channel path identifier**

Each channel path in the system is assigned a unique identifier value that is known as a *channel path identifier* (CHPID). A total of 256 CHPIDs are supported by a CSS, and a maximum of 1536 are supported per system (CPC).

The channel subsystem communicates with I/O devices through channel paths between the channel subsystem and control units. On IBM Z, a CHPID number is assigned to a physical location (slot/port) by the client, by using the hardware configuration definition (HCD) tool or input/output configuration program (IOCP).

## **Control units**

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

## **Multiple subchannel sets**

Do not confuse the multiple subchannel sets (MSS) functionality with multiple channel subsystems. In most cases, a *subchannel* represents an addressable device. For example, a disk control unit with 30 drives uses 30 subchannels for base addresses. An addressable device is associated with a device number, which is commonly (but incorrectly) known as the device address.

## **Subchannel numbers**

Subchannel numbers (including their implied path information to a device) are limited to four hexadecimal digits by the architecture (0x0000 to 0xFFFF). Four hexadecimal digits provide 64 KB addresses, which are known as a *subchannel set*.

IBM has reserved 256 subchannels, leaving over 63 KB subchannels for general use. Again, addresses, device numbers, and subchannels are often used as synonyms, although this is not technically accurate. You might hear or read that there is a *maximum of 63.75 KB addresses* or a maximum of 63.75 KB device numbers.

The processor architecture allows for sets of subchannels (addresses), with a current implementation of four sets. Each set provides 64 KB addresses. Subchannel set 0, the first set, reserves 256 subchannels for IBM use (as above) leaving 63.75 KB. Each of subchannel sets 1, 2 and 3 provides 64 K minus one subchannel. In principle, subchannels in either set can be used for any device-addressing purpose.

The additional subchannel sets, in effect, add an extra high-order digit (either 0, 1, 2 or 3) to existing device numbers. For example, you might think of an address as 08000 (subchannel set 0), 18000 (subchannel set 1), 28000 (subchannel set 2), or 38000 (subchannel set 3). Adding a digit is not done in system code or in messages because of the architectural requirement for four-digit addresses (device numbers or subchannels). However, certain messages contain the subchannel set number. You can mentally use that as a high-order digit for device numbers. Only a few requirements refer to the subchannel sets 1, 2 and 3 because they are only used for these special devices. JCL, messages, and programs rarely refer directly to these special devices.

Moving these special devices into an alternate subchannel set creates more space for device number growth. The appropriate subchannel set number must be included in the input/output configuration program (IOCP) definitions or in the HCD definitions that produce the input/output configuration data set (IOCDs). The subchannel set number defaults to zero.

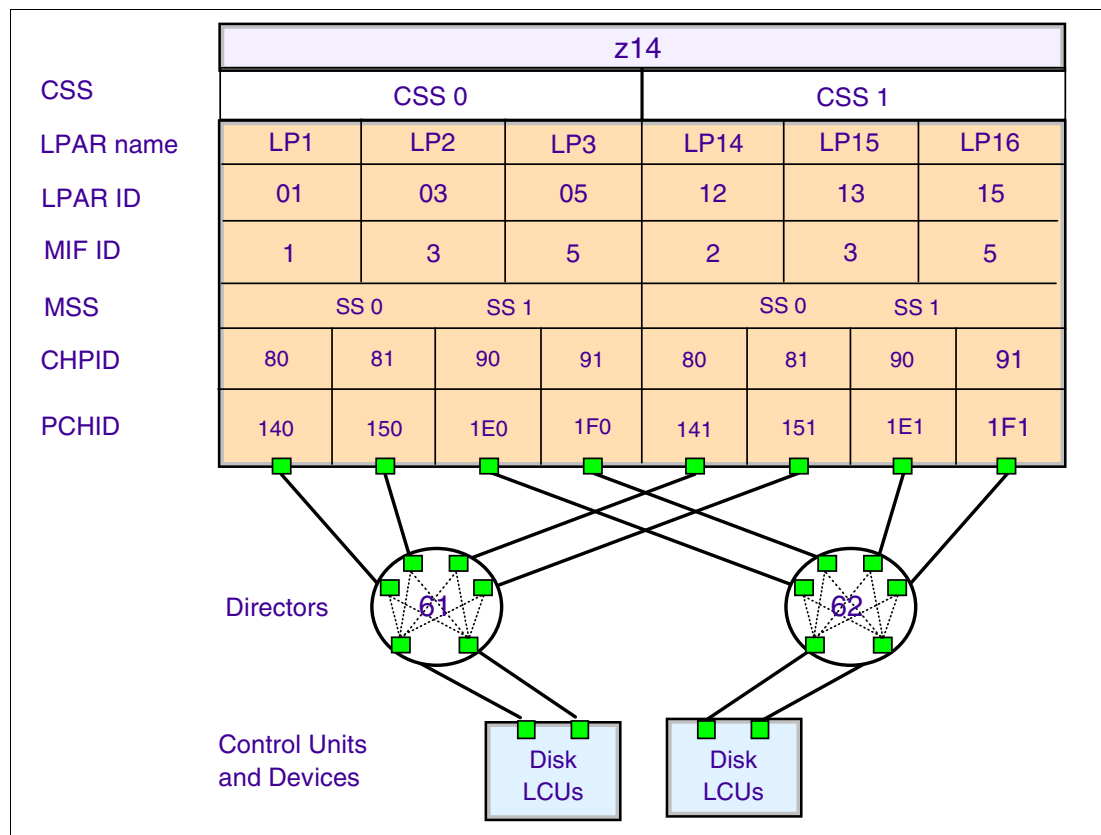


Figure 3-1 Channel subsystem

Figure 3-1 illustrates a number of important points for I/O connectivity:

- ▶ Two CSSs are shown; however, the z14 allows six CSSs.
- ▶ An LPAR is associated with a specific CSS. Also note that LPARs have unique names across the entire system.
- ▶ Multiple LPs (up to 15) may be associated with a CSS (except for CSS 5, which has 10).

- ▶ 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:** CHPID numbers are arbitrarily selected. For example, we could change CHPID 80 (in either or both CSSs in the illustration) to IBM C3 simply by changing a value in the IODF/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 (and different CSS). As shown in Figure 3-1 on page 110, PCHID 0140 is the first channel on the I/O features and PCHID 0141 is the second channel on the same I/O feature.

Figure 3-1 on page 110 also illustrates the relationship between LPs, CSSs, CHPIDs, and PCHIDs.

## 3.7 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 data set (IOCDs). The IOCDs is loaded into the Hardware System Area (HSA) and initialized during Power-on Reset.

The following tools are provided for the client to maintain and optimize the I/O configuration of your environment.

### Hardware Configuration Definition (HCD)

HCD supplies an interactive dialog to generate your IODF and subsequently your IOCDs. It is strongly recommended 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.

### Hardware Configuration Manager (HCM)

HCM is a PC-based graphical application 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.

### CHPID Mapping Tool (CMT)

The CMT provides a mechanism for customizing the CHPID assignments for an IBM Z server. The purpose is to avoid connecting critical paths to single points of failure. CMT is available for download from IBM Resource Link.

## 3.8 Displaying channel types

```

❑ Display CHPID Matrix

ISF031I CONSOLE PATRICK ACTIVATED
-D M=CHP
IEE174I 10.40.41 DISPLAY M 628
CHANNEL PATH STATUS
  0 1 2 3 4 5 6 7 8 9 A B C D E F
2 . . . . . + + + +
3 . . . . . + + + +
4 . . . . . + + + +
5 . . . . . + + + +
6 . . . . . + + + +
7 . . . . . + + + +
A + + + + + + + +
D . . . . . + + + +

***** SYMBOL EXPLANATIONS *****
+ ONLINE      @ PATH NOT VALIDATED  - OFFLINE  .
* MANAGED AND ONLINE  # MANAGED AND OFFLINE
CHANNEL PATH TYPE STATUS
  0 1 2 3 4 5 6 7 8 9 A B C D E F
2 00 00 00 00 00 00 00 00 00 00 00 00 1A 1A 1A 1A
3 00 00 00 00 1A 1A 00 00 00 00 00 00 00 1A 1A 1A 1A
4 00 00 00 00 1B 1B 00 00 1B 1B 1B 1B 00 00 00 00 00
5 00 00 00 00 1B 1B 00 00 1B 1B 1B 1B 00 00 00 00 00
6 00 00 00 00 00 00 00 00 00 00 00 00 00 1A 1A 1A 1A
7 00 00 00 00 00 00 00 00 00 00 00 00 00 1A 1A 1A 1A
A 14 11 11 11 11 11 11 10 00 00 00 00 00 00 00 00 00
D 00 00 00 00 00 00 00 00 00 00 00 11 11 00 00 00 00

***** SYMBOL EXPLANATIONS*****
00 UNKNOWN
10 OSA EXPRESS
11 OSA DIRECT EXPRESS
12 OPEN SYSTEMS ADAPTER
14 OSA CONSOLE
16 CLUSTER BUS SENDER
17 CLUSTER BUS RECEIVER
18 INTERNAL COUPLING SENDER
19 INTERNAL COUPLING RECEIVER
1A FICON POINT TO POINT
1B FICON SWITCHED
1D FICON INCOMPLETE
1E DIRECT SYSTEM DEVICE
1F EMULATED I/O
20 RESERVED
21 INTEGRATED CLUSTER BUS PEER
22 COUPLING FACILITY PEER
23 INTERNAL COUPLING PEER
24 INTERNAL QUEUED DIRECT COMM
25 FCP CHANNEL
26 COUPLING OVER INFINIBAND
32 UNKNOWN
33 COUPLING OVER PCIE
NA INFORMATION NOT AVAILABLE

❑ CHPID 6C = FICON Channel Type

```

Figure 3-2 Display CHPID Matrix command to display channel types

### Displaying channel types

Figure 3-2 shows some of the output from the M=CHP command z/OS command (it has been edited for documentation purposes). The symbol explanations shown on the right side of the figure relate to the information shown on the left on the operator console.

This z/OS 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:** Whereas the first section displays the status of channels available to the z/OS image only, the second section provides information about *all* channels installed on the server.



## Virtualization and Logical Partition (PR/SM) concepts

This chapter explains virtualization concepts in general and in detail, one type of virtualization in the z14 CPC, that is, PR/SM, also known as Logical Partitioning (LPAR).

Along these explanations, other IBM virtualization implementations are lightly described, such as z/VM, PKM for IBM Z, PowerVM®, and VTS.

In the PR/SM part, we cover the concept of a logical partition (LP), and how to define it in a z14 CPC. Here, we use the acronym PR/SM to describe the Licensed Internal Code (LIC) software that partitions the CPC, creating the virtualized LPs. LIC means a type of internal software code not perceived by the customer.

PR/SM allows a system programmer to allocate CPC hardware resources (including PUs, CPC memory, and I/O channels) among LPs.

Then, in PR/SM mode, the resources of a CPC can be distributed among multiple control programs (operating systems) that can run on the same CPC simultaneously. Each control program has the use of resources defined to the LP in which it runs.

## 4.1 Virtualization definitions and properties

Let's cover some virtualization concepts.

### Virtualization definitions

Virtualization is a 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. In a sense, virtualization is a synonym for resource sharing and consequently increases the resource utilization. Virtualization is still a key strength of Z platforms, and useful for cloud implementation.

- ▶ A definition: Virtualization is the ability for a computer system to share resources so that one physical server can act as many virtual servers, or virtual machines (VM). This can reduce the number of processors and hardware devices needed.
- ▶ Another virtualization definition can be this one: “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.”
- ▶ Another definition: “Virtualization is the technique that adds a logical layer for the real hardware, separating the Virtual Resources from the actual physical resources. This technique allows a better hardware exploitation, allowing a better sharing and of resources, reducing considerably the amount equipment. Also, the Virtualization, increases the agility, flexibility and scalability of system, giving as a result more productivity, efficiency and responsiveness to the business requirements.”

### Virtualization properties

Creating many VMs consisting of virtualized processors, communications, memory, and I/O devices, can reduce acquisition 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.

Benefits of CPC virtualization, such as improved utilization, flexibility, responsiveness, workload mobility and readiness for disaster recovery.

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.

Then, virtualization 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 PUs into less and more powerful ones.



## 4.2 Virtualization concepts

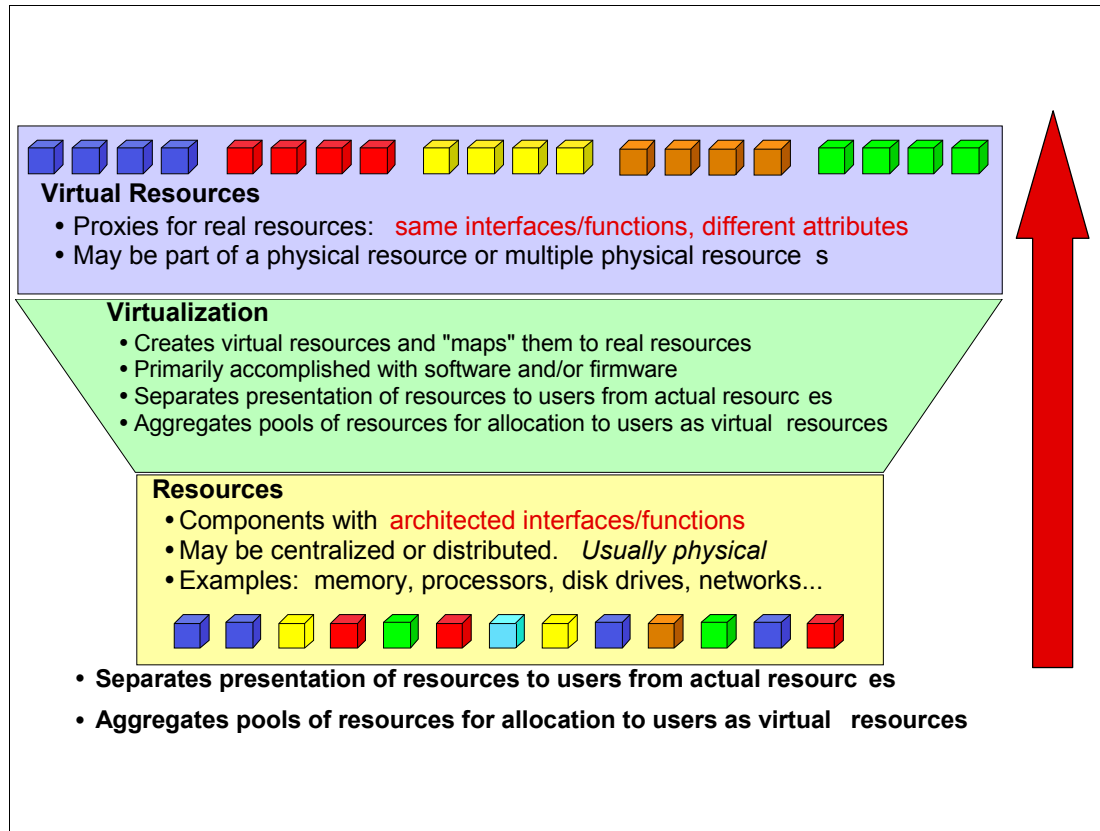


Figure 4-1 Virtualization concepts

### Virtualization benefits

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 shared resources, due to increasing management.

## 4.3 Virtualized physical resources

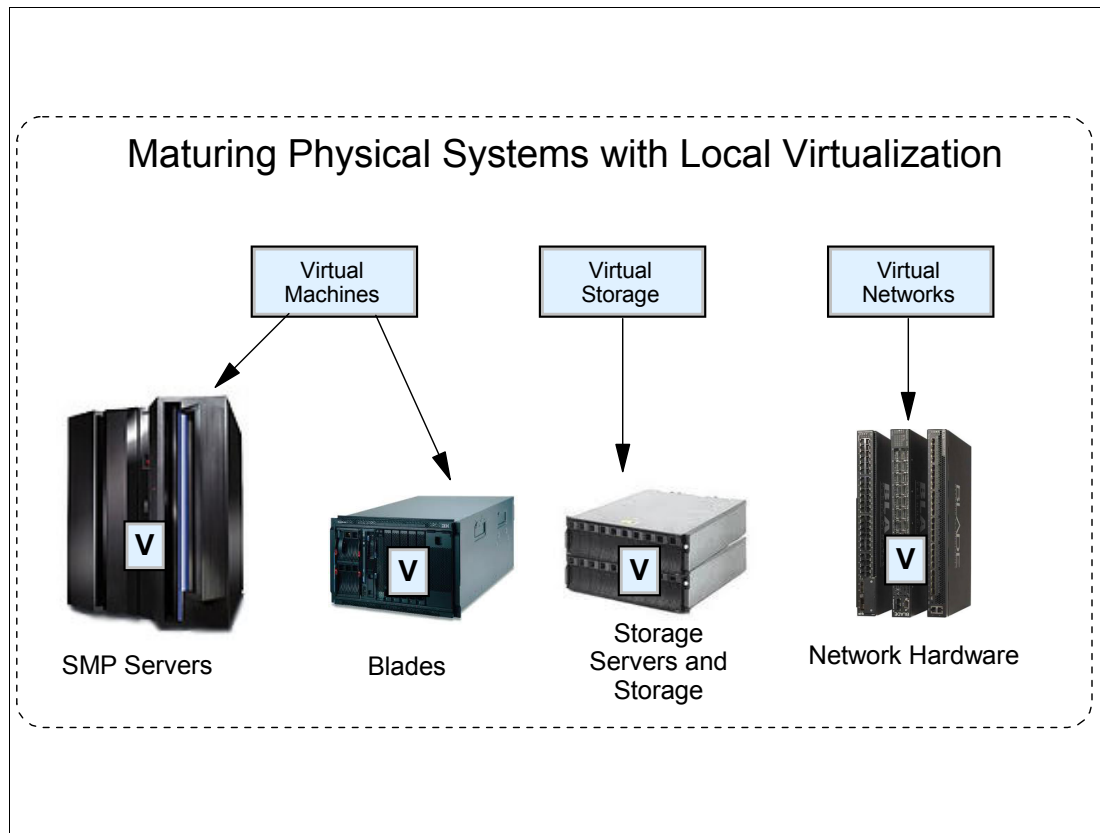


Figure 4-2 Virtualized physical resources

### Virtualized physical resources

Primarily virtualization logic is accomplished by software (hypervisors such as IBM z/VM, PowerVM) and/or LIC (the IBM VTS tape virtualizes). PR/SM is an example of a hypervisor that is a LIC.

In a sense, virtual resources are proxies for real resources (as the ones pictured in Figure 4-2) with the same architecture interfaces and functions, but different attributes.

The resources that may be virtualized are:

- ▶ CPC memory. When CPC memory 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 CPC memory. 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 PU 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 CPC memory resource. By virtualizing CPC memory, the programs have the illusion of having much more addresses in CPC memory than bytes in reality. The overflow is moved to storage in page data sets. Then, more programs can fit together, causing a higher multiprogramming level, and more PU utilization and more I/O operations.

It is important to note that CPC memory virtualization has an operating system scope, that is, the virtualization is done among the programs running within such an operating system

copy. Among different operating systems (LPs) in the same CPC, the CPC memory is sliced among them, but with the possibility to reconfigure manually from one image to the other.

- ▶ **Processors.** They are virtualized by hypervisors running underneath operating system images in the same CPC. An example can be PR/SM (a hypervisor) delivering physical PUs to a z/OS running in an image created by PR/SM. In other words, the physical PUs are shared between copies of operating system images. If we have a peak on demand for PUs in one image, automatically the hypervisor delivers the majority of physical PUs to that image. If the peak moves away to another image, the same behavior is repeated. However, if the adding of all PU demand is greater than the total physical PU capacity, then the installation must deliver a criteria (weights) indicating the relative importance of each image. It is also possible to dedicate some PUs to certain images.
- ▶ **I/O channels.** They can be shared (a sort of virtualization) automatically among images of the same CPC under an PR/SM or z/VM. The same FICON channel and the z14 PCIe Hyperlink 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, during channel virtualization there is a total emulation of the hardware, as in the case of Hype Sockets 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 CPC memory 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.

## 4.4 Hypervisor types

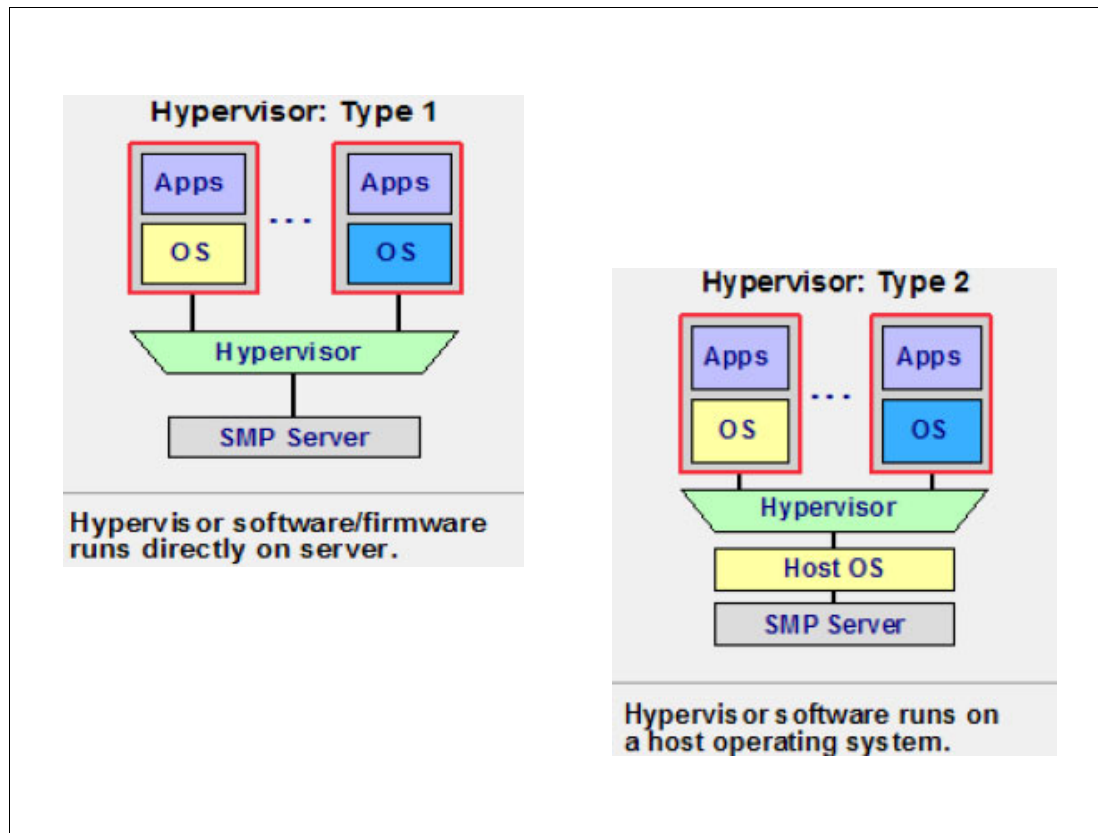


Figure 4-3 Hypervisor types

### Hypervisor types

A hypervisor is a sort of operating system. Its major role is to virtualize physical resources such as processors, CPU memory 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 4-3 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 labeled OS are the virtual machines (guest OS). z/VM and PR/SM are examples of a Type-1 hypervisor. Some other examples are the XEN Open Source 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 4-3 illustrates a Type-2 hypervisor. The box labeled Host OS is the host operating system. The box labeled 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.

## 4.5 Hypervisor technologies (I)

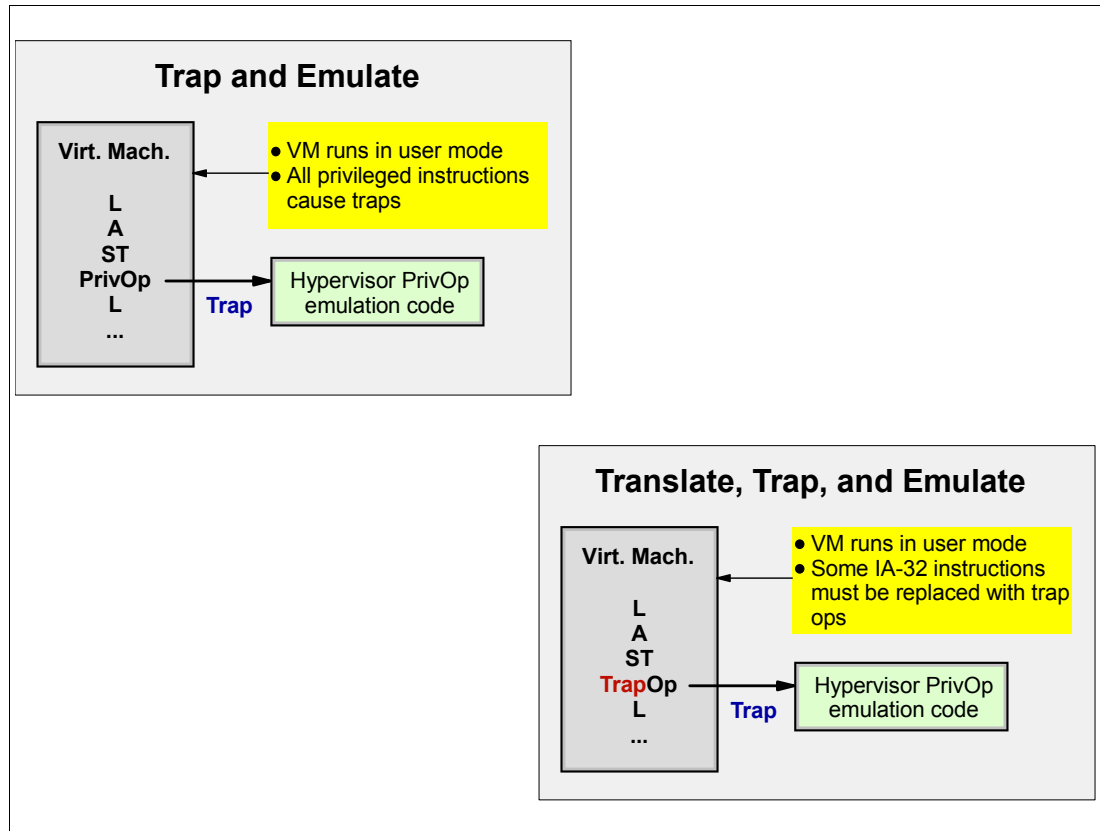


Figure 4-4 Hypervisor technologies

### 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 4-4 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 4-4. Examples are VMWare and Microsoft VS.

## 4.6 Hypervisor technologies (II)

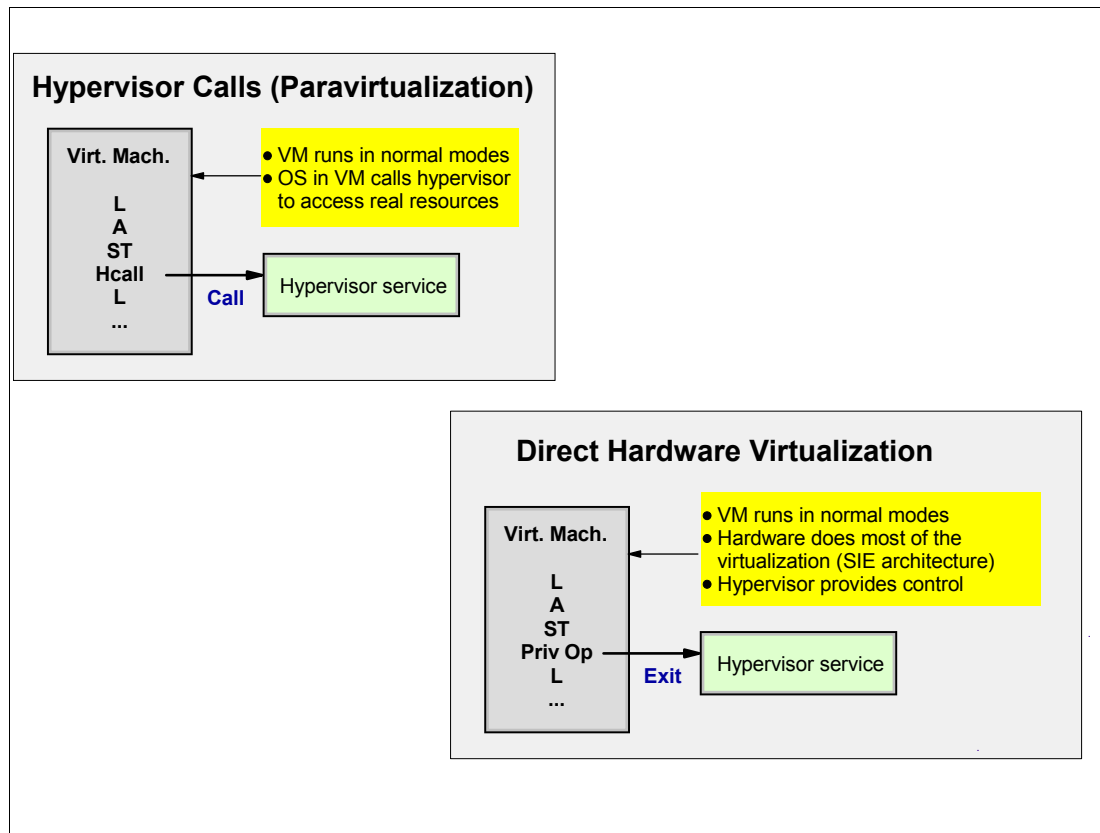


Figure 4-5 Hypervisor technologies (II)

### 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 4-5. The guest operating system kernel (for example, IBM 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 4-5, is used by the IBM Z (PR/SM and z/VM) family of mainframes.



## 4.7 IBM Hypervisors on IBM Z family

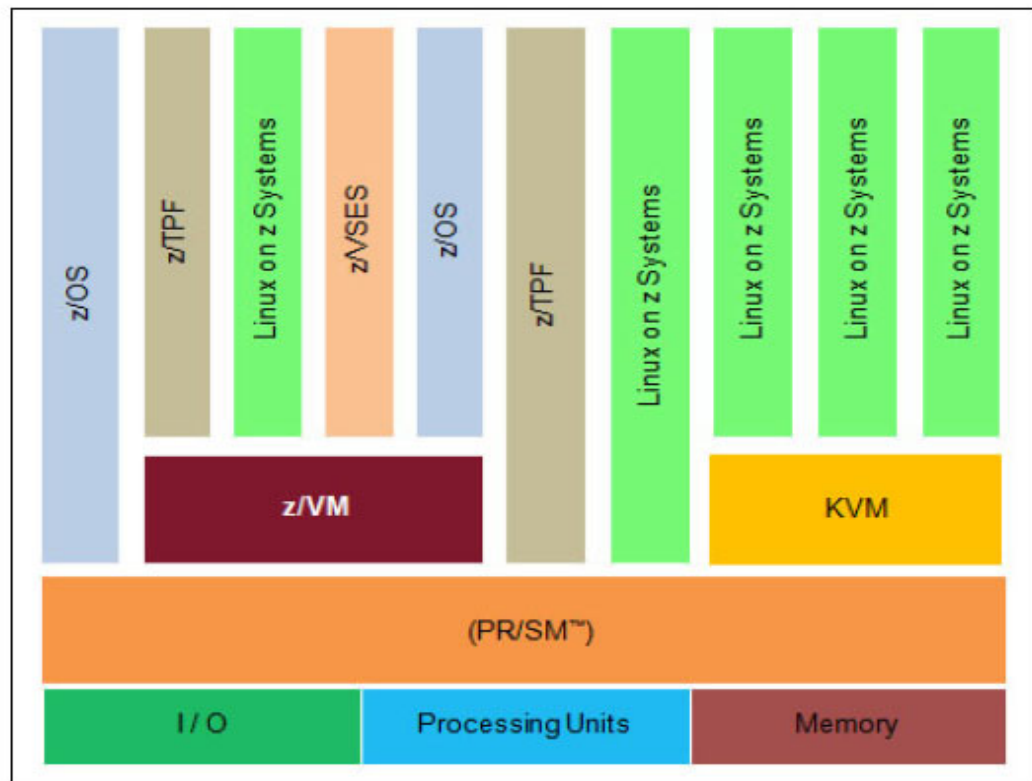


Figure 4-6 IBM Hypervisors on Z

### IBM Hypervisors on Z

The IBM hypervisors are PR/SM, z/VM, PowerVM, and KVM for Z. Starting with z10 it is not possible to have a Z CPC without PR/SM being active. Then, z/VM must run under PR/SM, even though it is defined with just one LP.

PR/SM is an LIC (a software unseen by the customer) hypervisor is extensively covered in this chapter starting at “CPC in PR/SM mode” on page 128.

The various virtualization options in Z platforms allow you to build flexible virtualized environments to take advantage even of open source software (such as KVM for IBM Z) or upgrade to new cloud service offerings, such as infrastructure as a service (IaaS) and platform as a service (PaaS).

## 4.8 z/Virtual Machine (z/VM)

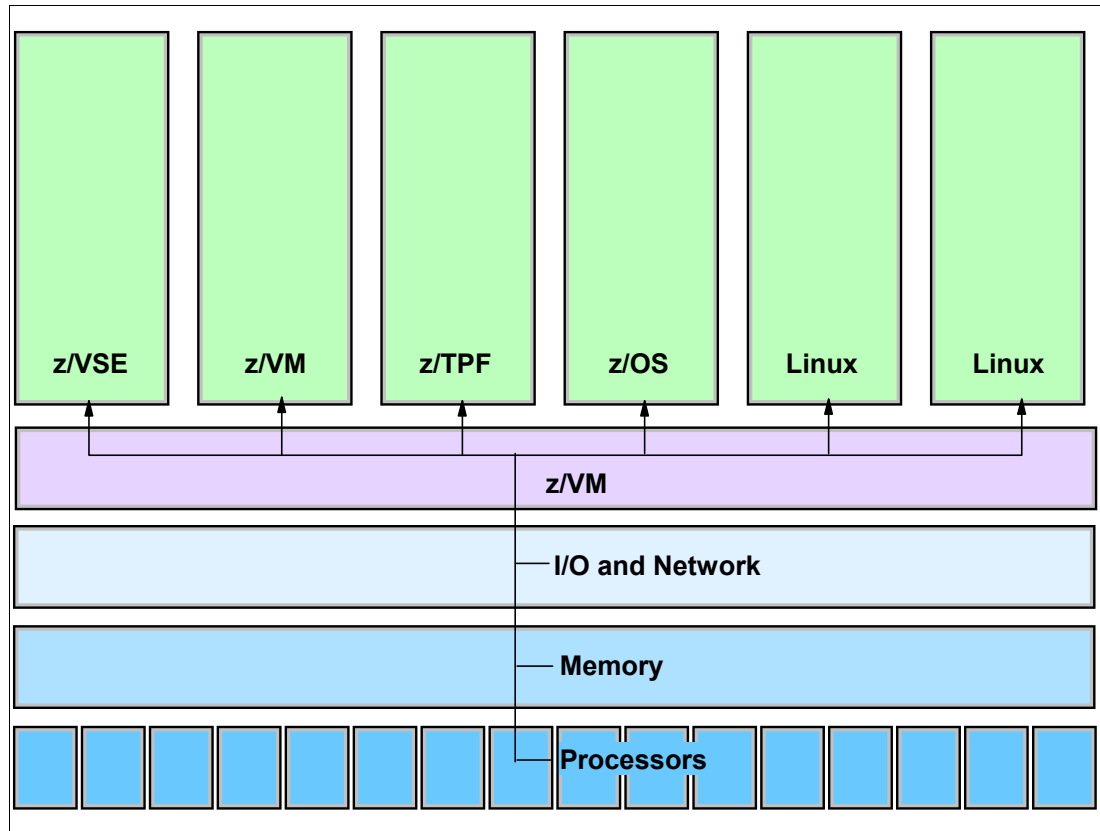


Figure 4-7 z/Virtual machine (z/VM)

### **z/Virtual machine (z/VM)**

z/VM is a chargeable type-1 hypervisor allowing the sharing of the MF's physical resources such as processors, memory, network adapters, and disks. All these resources are managed by z/VM. Typically, the z/VM hypervisor is used to run Linux virtual machines (in this case the PU type is IFL), but other OS, such as z/OS (in this case the PU type is CP) can also run on z/VM. z/VM is a proven and well established virtualization platform. It provides industry-leading capabilities to efficiently scale both horizontally and vertically.

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 hypervisor 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 device number of the tape drive may or may not be the same as the real device number of the tape drive. Therefore, a virtual machine only knows "virtual hardware" that may or may not exist in the real world. Figure 4-7 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 in to 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 or virtual machines executing Z operating systems such as: z/OS, z/TPF, Linux, z/VSE, CMS, or z/VM (forming in this case, a cascade).

CP (as PR/SM) 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. Observe that the maximum number of PR/SM logical partitions in a CPC is only 85.

## 4.9 z/VM options in HMC

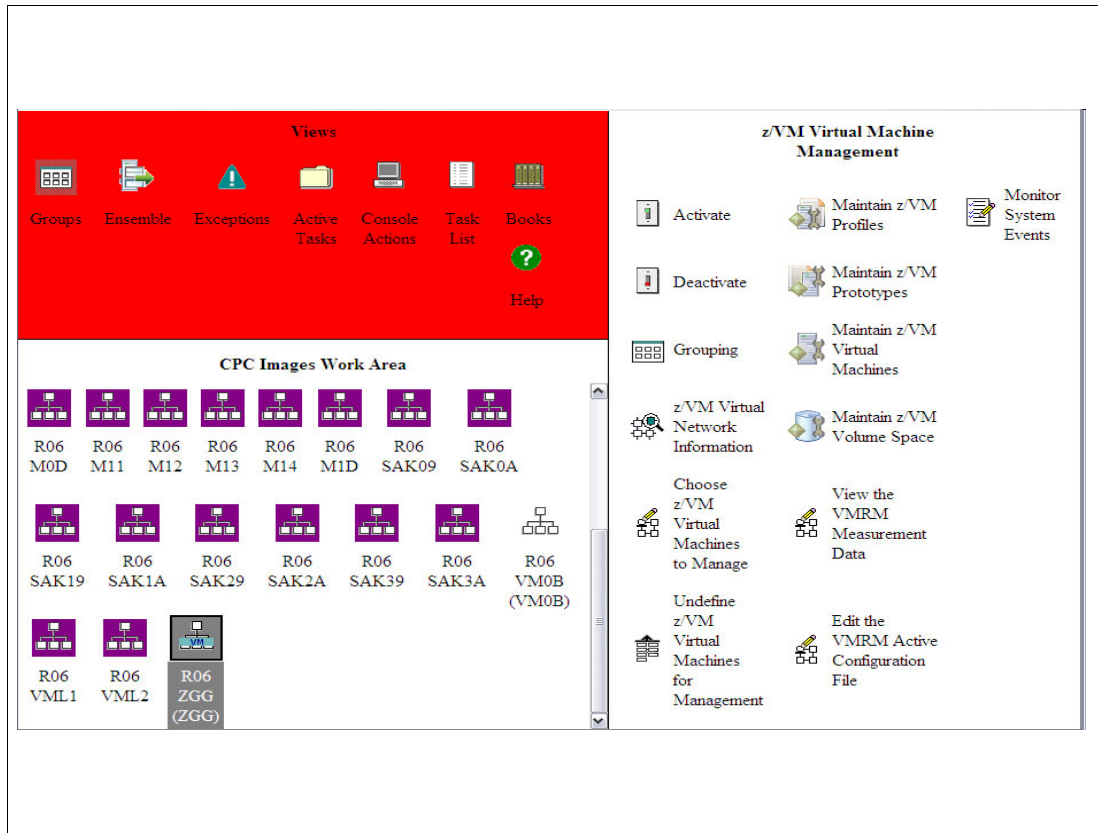


Figure 4-8 z/VM main panel in HMC

### z/VM options in HMC

Allows basic z/VM functions to be performed from HMC, the z14 hardware console. Starting with z/VM 5.3, no network connection is required.

This function uses a 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

## 4.10 KVM for IBM Z and DPM

Here, we describe the hypervisor KVM for IBM Z and the Dynamic Partition Manager (DPM).

### KVM for IBM Z description

It is an open source virtualization solution, that provides simple, cost-effective server virtualization for Linux workloads running on the Z platform. It is a Type-1 hypervisor that delivers server virtualization based on open source KVM Linux technology.

The KVM hypervisor for IBM Z enables you to share real PUs (he IFLs), memory, and I/O resources through platform virtualization. It can coexist with z/VM virtualization environments, Linux on IBM Z, z/OS, z/VSE, and z/TPF.

The KVM hypervisor for IBM Z is optimized for scalability, performance, security, and resiliency, also providing standard Linux and KVM interfaces for simplified operational control.

The KVM hypervisor for Z for the z14 is offered with the following Linux distributions:

- ▶ SUSE Linux Enterprise Server 12 SP2 (or higher) with service
- ▶ Canonical Ubuntu 16.04 LTS (or higher) with service.

### Dynamic Partition Manager (DPM)

DPM is a management infrastructure mode in the z14. It is intended to simplify virtualization management and is easy to use, especially for those who have less experience with Z. It does not require you to learn complex syntax or command structures.

DPM provides simplified hardware and virtual infrastructure management, including partition lifecycle and integrated dynamic I/O and PCIe functions management for Linux running under PR/SM, under KVM for IBM Z, and under z/VM 6.4. Using DPM, an environment can be created, provisioned, modified without disrupting running workloads, and monitored for troubleshooting.

### 4.10.1 Hypervisor terminology

Here, we show the different terms used in the three IBM hypervisors, as follows:

- ▶ **PR/SM**: Logical Partition, Logical CP, PR/SM, Weight
- ▶ **z/VM**: Virtual Machine, Virtual CP, Control Program, Weight
- ▶ **PowerVM** (not for Z): Virtual server, Virtual Processor, PowerVM Hypervisor, Entitlement Capacity

Explaining:

- ▶ 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.

## 4.11 CPC in PR/SM mode

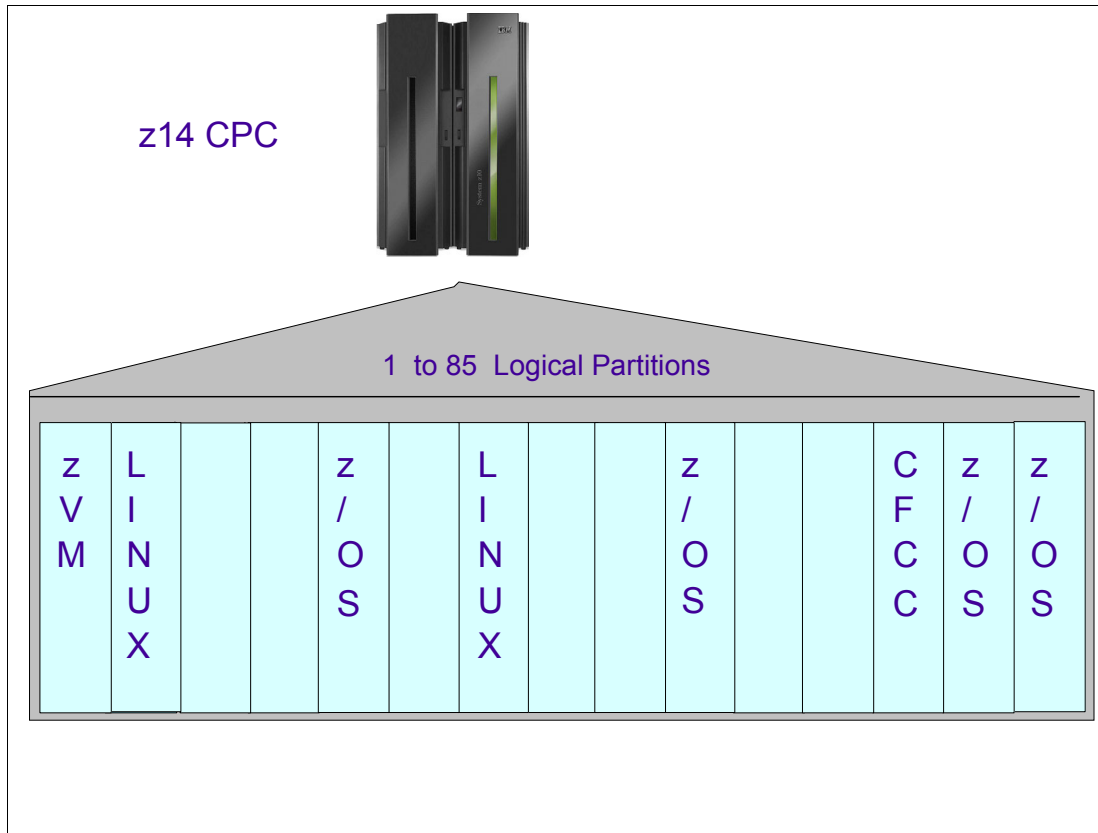


Figure 4-9 CPC in PR/SM mode

### Processor Resource/System Manager (PR/SM)

PR/SM is a standard mandatory free of charge feature of z14 CPC. It is formed by:

- ▶ Multi High Performance Guest Support (MHPGS) in VM  
This allows several preferred guests under z/VM, all with performance close to that available when running native. This function was long ago incorporated with the z/VM control program.
- ▶ Logical Partitioning (LPAR)  
It is a software LIC located in Hardware System Area (HSA). It allows a CPC to be divided into multiple logical partitions (LP), that is, virtualization. This capability was designed to help with isolation of 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.

Usually the terms LPAR and PR/SM are used interchangeably, both meaning Logical Partitioning.

An LP has the following properties:

- ▶ Each LP is a set of physical resources (processor, memory, and channels) controlled by just one independent image of an operating system, such as z/OS, Linux, CFCC, z/VM, z/TPF, or z/VSE.
- ▶ You can have up to 85 LPs in a z14 CPC.

- ▶ Each LP is defined through IOCP/HCD. For example:
  - RESOURCE PARTITION = ((LP1,1),(LP2,2)) statement defines two LPs, each one with an LPAR name and a MIF ID)

Through the concept of pre-defined 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 PR/SM capping is to be used for this LP, the LP memory size, use of virtual flash memory (VFM), 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.
- ▶ FICON channels and PCIe HyperLink channels can be dedicated, reconfigurable (dedicated to one LP, but able to be switched manually between LPs), or shared.
- ▶ The CPC memory and the VFM used by an LP is dedicated, but can be manually reconfigured from one LP to another with prior planning and without stopping (no IPLing) both LPs.
- ▶ Although it is not strictly accurate, most people use the terms PR/SM and LPAR interchangeably to refer to the LIC 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.

The majority of the z/OS code does not interact directly with the LPARs that run in the system.

## 4.12 Logical PU concept

### Logical PU

A logical PU is a PU, as perceived by the OS running in the LP. To use an analogy, z/OS dispatches tasks (on logical PUs) and PR/SM dispatches logical PUs (on physical PUs). There are dedicated and shared (recommended) logical PUs.

Each logical PU in an LP is mapped into a State Descriptor Control Block (SDCB). SDCB contains the last saved state (PSW, registers, CP Timer) of such a logical PU. When ready, the logical PU has its SDCB in the PR/SM ready queue of such a PU pool type (set of logical PUs with the same personality, such as zIIP or CP). The ready queue is ordered indirectly by the LP Weigh value.

PR/SM's major function is to dispatch its running physical PU into a logical PU of the same logical pool type (CP, zIIP, IFL, ICF) through the use of the SIE instruction. z/VM Control Program uses SIE to dispatch a virtual PU in a physical PU.

The logic of SIE is to load the state of logical PU in the physical and forces physical to start instruction execution from this state.

A logical CP can only be dispatched in a physical CP, and so on. A shared logical PU has three states:

- ▶ Active, when running in the physical PU (also true for dedicated logical PU).
- ▶ Ready (or Suspended), when delayed by the lack of an available physical PU. The SDCB is in the PR/SM ready queue. There is one ready queue per pool of logical PUs. A dedicated logical PU never assumes such a state.

- ▶ Wait, set by the z/OS running in the LP because of a no work condition (also true for dedicated logical PU).

Intercept is the separation between logical and physical PU, state of physical PU is saved in the SDCB (including CP Timer). Reasons for Intercept:

- ▶ Voluntary wait (z/OS voluntarily gives up the processor by switching the PSW Wait bit 14 to ON), because there is nothing in a "Ready" state in the dispatchable unit access list (either a task control block or a service request block).
- ▶ End of an LPAR Time Slice, set by PR/SM to avoid a logical PU's dominance of the physical PU. At SIE moment, the LPAR decides dynamically such a value.



## 4.13 Shared and dedicated logical PU example

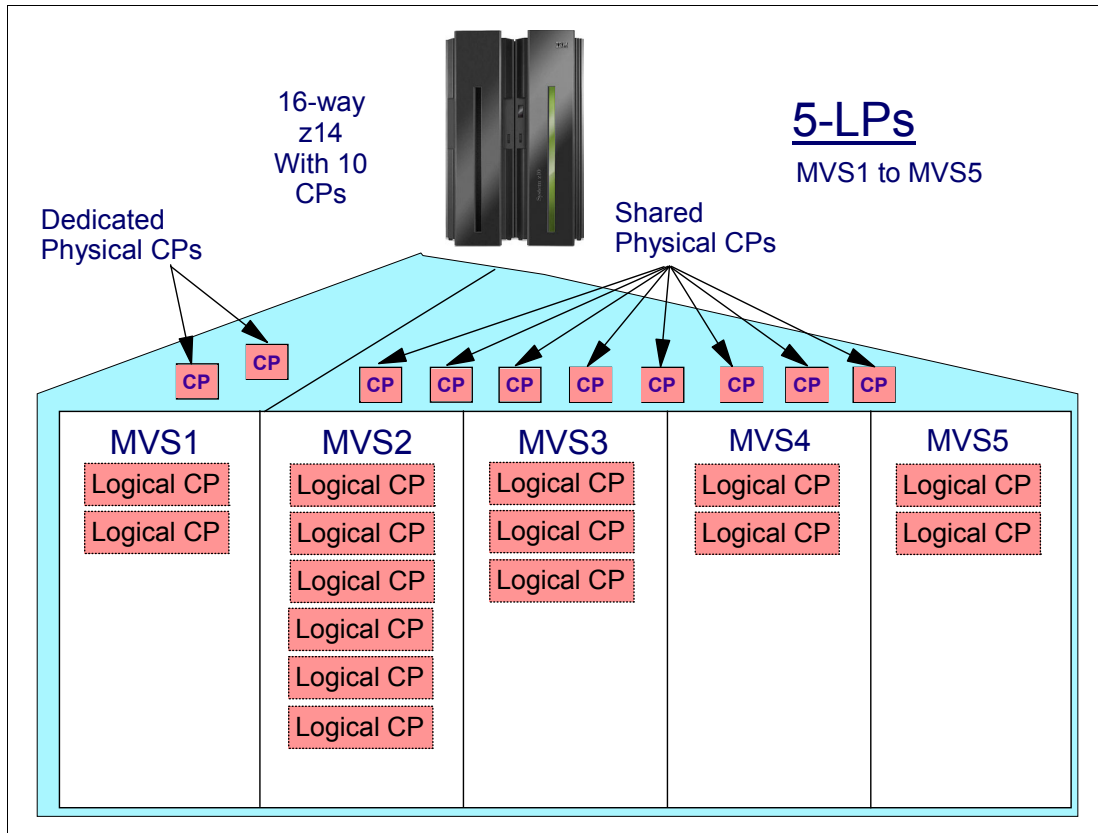


Figure 4-10 Shared and dedicated CPs example

### Logical and physical PUs

Here Physical PUs are understood as CP and zIIPs. The physical PUs can be dedicated or shared, as shown by the 10 PUs in Figure 4-10. If *dedicated*, the physical PU is permanently assigned to the same logical PU of just one LP. The advantage of this is less PR/SM overhead, and when the physical PU is required by the logical, the physical is always available. The PR/SM overhead increases in line with the proportion of logical shared PUs.

However, with sharing you have the ability of utilize the PU capacity that is not required by another sharing LP (that is in wait state). Normally, when a z/OS system that is using a shared PU goes into a wait, PR/SM releases the physical PU, which can then be used by another logical PU in ready state.

There are a number of parameters, that let you control the distribution of shared PUs between LPs. It is not possible to have a single LP use both shared and dedicated CPs, with the exception of the coupling facility LP.

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. PR/SM also continues to be used to run in different environments, such as system programmer test, development, quality assurance, and production, all in the same CPC.

## PR/SM PU Management

PR/SM management of the shared PU environment is described here.

Figure 4-10 on page 131 shows a 10-CP configured IBM z14 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 such figure, 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.

All the ideas described here apply also to zIIP, not only to CP in one z/OS LP.

## 4.14 PR/SM dispatching and shared CPs

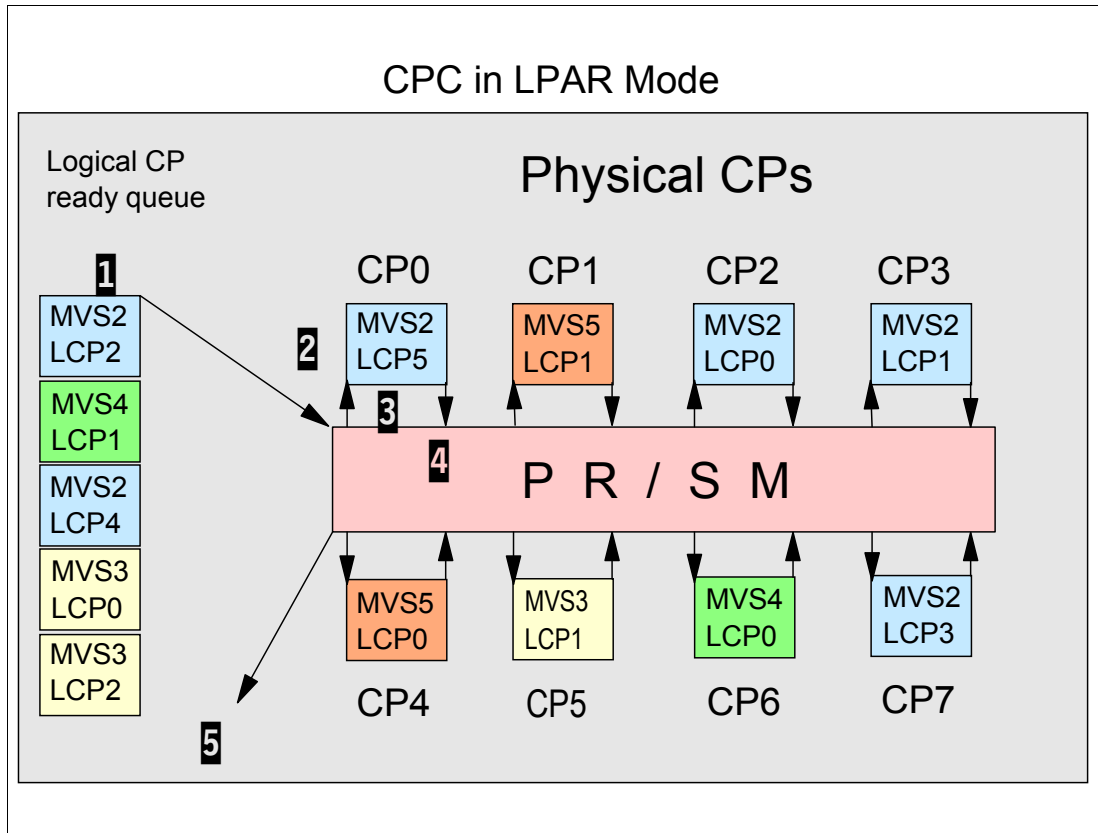


Figure 4-11 PR/SM logical CP dispatching

### PR/SM logical CP dispatching

PR/SM logic executes on all of the physical CPs. Then, PR/SM dispatches a logical CP on a physical CP (the one the PR/SM is currently running on) by issuing the SIE instruction, with the logical CP represented by a SDCB. There is one SDCB per each logical CP.

The logical CP is dispatched on the physical CP by copying the LP's logical CP status (PSW, registers, CP Timer) from the specific SDCB to the corresponding actual PU entities. This causes the z/OS code or application code in that LP to execute on the physical CP, through the logical CP.

Later, 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 PR/SM is automatically dispatched on the physical CP again. This PR/SM code, then chooses another logical CP to dispatch, and the whole process starts all over.

Figure 4-11 illustrates the flow of execution in a CPC, where we have eight shared physical CPs. Let us examine this now.

### Logical CP dispatching

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 PR/SM has up to 13 SDCBs to manage in our environment.

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 is a logical CP taken from the ready queue to executing on a physical CP? As mentioned before, PR/SM is responsible for dispatching a logical CP on a physical CP. PR/SM executes on each physical CP. When a logical CP is found ready, it is dispatched in a physical CP, by PR/SM issuing a SIE instruction, which switches the PR/SM code from the physical CP and replaces it with the code that was previously running on the logical CP.

The steps that occur in dispatching a logical CP, illustrated in Figure 4-11 on page 133, are as follows:

- 1** - The next logical CP to be dispatched is chosen from the logical CP ready queue based on its LP weight.
- 2** - PR/SM dispatches the selected logical CP (LCP5 of MVS2 LP) on a physical CP0.
- 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 causing an intercept, or it enters a voluntary wait (lack of service in MVS2).
- 4** - As shown, the logical CP keeps running (for example) until it uses all its time slice. At this point the logical CP5 (of MVS2) state is saved in its SDCB and control is passed back to PR/SM LIC, which restarts execution on physical CP0 again.
- 5** - PR/SM determines why the logical CP was intercepted and requeues the logical CP accordingly. If it is still ready (no wait), it is requeued on the logical CP ready queue and life goes on, that is, back to step **1**. If no more SDCBs are in the ready queue, PR/SM places the physical CP in wait state.

This process occurs recursively with each physical CP.

## 4.15 PR/SM Weights

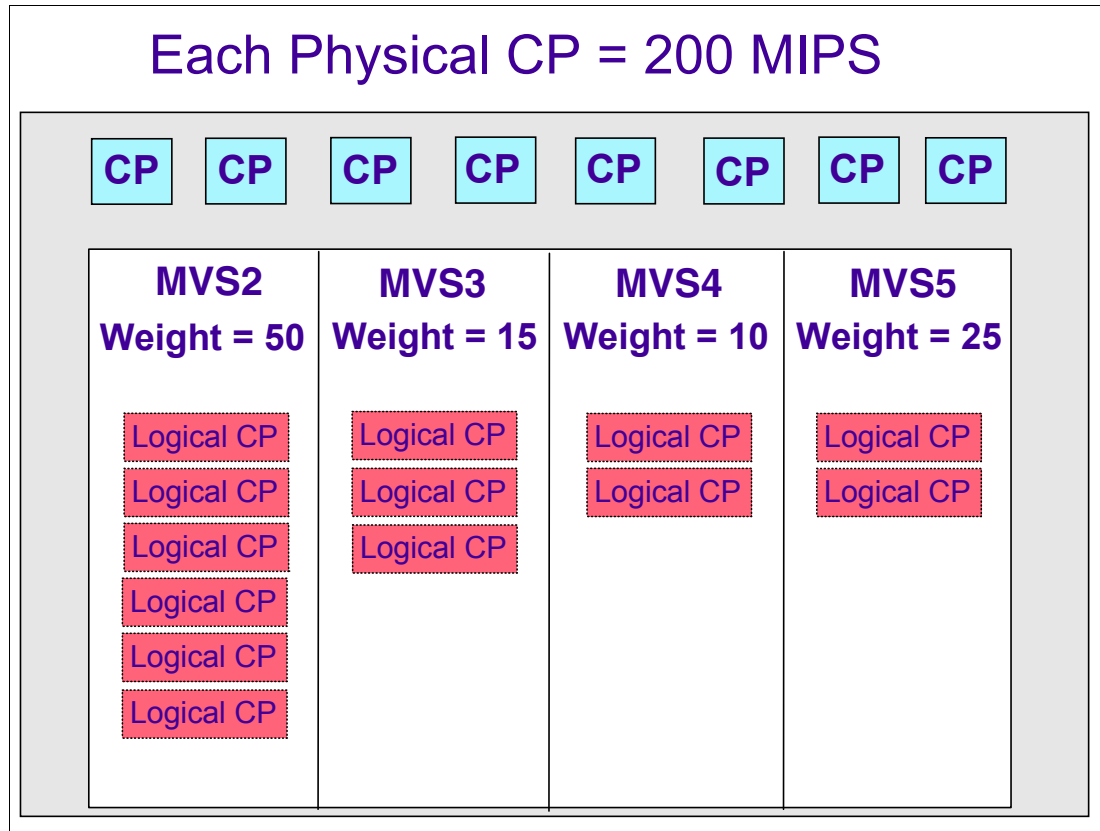


Figure 4-12 PR/SM weights

### PR/SM weights

PR/SM weights are used to control the distribution of shared physical PUs between logical PUs of LPs. Therefore, for LPs with dedicated PUs, PR/SM weights are not assigned.

PR/SM weights determine the guaranteed (minimum) amount of physical PU 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 CP-bound).
- ▶ The LP is capped using traditional PR/SM capping.

An LP may use less than the guaranteed amount if it does not have such workload to execute. Similarly, it can use more than its weight, if the other LPs are not using their guaranteed amount.

PR/SM LIC uses *weights* and the *number of logical CPs* to decide the priority of logical PUs in the logical PU ready queue. The following formulas are used by PR/SM in the process of controlling the dispatching of logical PUs:

- ▶  $WEIGHT(LP_x)\% = 100 * WEIGHT LP_x / \text{SUM\_of\_ACTIVE LPs WEIGHTs}$
- ▶  $TARGET(LP_x) = WEIGHT(LP_x)\% * (\# \text{ of NON\_DEDICATE\_PHYS\_PUs})$

## Formula definitions

The definitions of the formula terms are:

- WEIGHT(LPx)%** Indicates the percentage of the total shared physical PU capacity that is guaranteed to this LP. This percentage varies depending on the weights of all the active LPs. In the example, the MVS2 value is 50%, assuming all the LPs in the CPC are active.
- TARGET(LPx)** Indicates, in units of shared physical PUs, how much PU resource is guaranteed to the LP. This figure cannot be greater than the number of logical PUs in the LP, because you cannot use more physical PUs than the number of logical PUs you have defined. By the way, each logical PU can be dispatched on only one physical PU at a time. So, even if there are eight physical PUs available, an LP that has been defined with only four logical PUs can only ever use four of the physical PUs at one time.

It is important to remember that all these considerations apply to any type of PU processor, at a z/OS LP, for zIIP and CP. Therefore, in an PR/SM profile the customer should define the number of logical zIIPs, together with their respective weights.

## PU Capping

*PU Capping* is process used to artificially limit the PU consumption rate of a specific set of dispatchable units (as TCBs and SRBs), usually associated with a user transactions workload.

You may cap the PU resource with the granularity of an LP, **or** of a WLM service class with a Parallel Sysplex scope.

## Major reasons for capping

Here, we present reasons why an enterprise should cap the PU resource:

- ▶ The LP software products are charged on consumed MSU/h basis.  
The consumed MSU/h per month is calculated by the peak hour of the month. To avoid a high peak hour, many customers cap such LPs.
- ▶ Enforcing Service Bureau limits on customer's workload.  
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 policy reasons.  
Following a hardware PU 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 due to PU cache degradation.  
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 reverting the process.

- ▶ 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.

## Types of capping

We have the following types of capping in the mainframe platform:

- ▶ PR/SM capping, where the full LP is capped. Refer to “PR/SM capping” on page 137. The intelligence and the capping executor are located in the PR/SM hypervisor:
  - The limit for capping is established by the LP weights (for zIIP and CP) at 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) in the CPC.
- ▶ WLM Resource Group capping, where just a set of WLM service classes (a set of transactions sharing the same WLM goal) 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.
- ▶ Defined capacity or soft capping, where the full LP is capped. The limit is indicated in MSU/h in the HMC PR/SM profile. The intelligence is located in WLM and the executor is in the PR/SM hypervisor.

The next sections describe these types of capping.

## PR/SM capping

PR/SM capping is a function used to ensure that an LP’s use of the physical PU cannot exceed the amount specified in its Target(LP<sub>x</sub>). PR/SM capping is set at the HMC Image Profile.

Normally, an LP can get more PU resources than the amount guaranteed by its Target(LP<sub>x</sub>); in “PR/SM Weights” on page 135, we discuss how to calculate the Target(LP<sub>x</sub>) 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(LP<sub>x</sub>) of the remaining active LPs is recalculated.

If you want to prevent an LP from ever being able to use more than its Target(LP<sub>x</sub>), even if there is spare PU resource available, you would use the PR/SM capping feature. PR/SM capping is implemented by the PR/SM 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(LCP<sub>x</sub>).

At frequent intervals (every few seconds or so), PR/SM compares the Target(LCP<sub>x</sub>) 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, PR/SM 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.

## 4.16 PR/SM capped versus uncapped

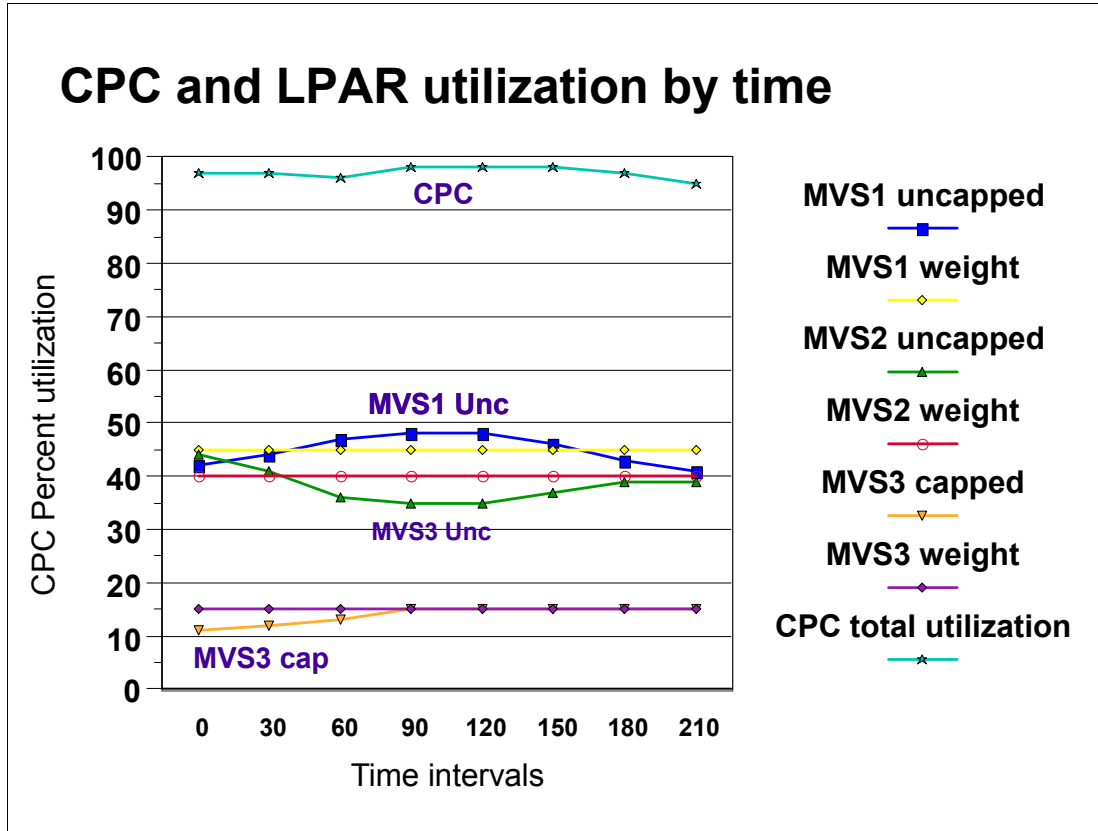


Figure 4-13 Capped and uncapped LPs

### Capped and uncapped LPs

Figure 4-13 illustrates what happens when there is spare CP resource and the CPC 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 CPC resource. For example, a weight of 45 equals 45% of the total CPC.

Each LP's weight is shown as a separate straight horizontal line in the chart. The total CPC utilization is shown as a separate line close to the top of the chart.

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 PR/SM shared CP handling.

### **WLM Resource Group (RG) capping**

WLM implements capping in its defined transactions workloads as well, with added flexibility, when compared with PR/SM capping.

About WLM RG capping granularity, you can cap a set of service classes within a sysplex. With PR/SM 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 more details about this subject.

## 4.17 Defined capacity or Soft capping

- Sum of defined capacity as a basis for charge
- Ability to handle workload spikes

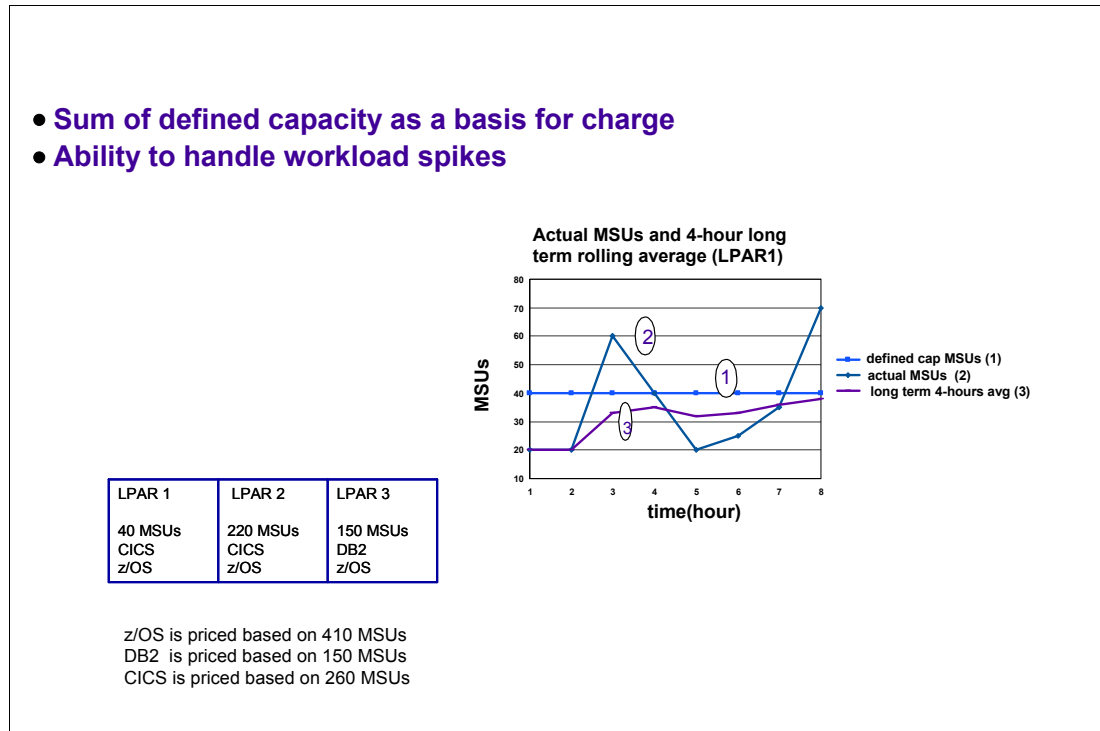


Figure 4-14 Defined capacity capping

### Defined capacity capping

Workload charging (WLC) is an IBM contract dealing with software fees. With WLC, the software products cost is based on the consumption of the LP running the product, measured in a 4-hour rolling average MSU/hour (captured during 5-minute intervals). Along this rolling average the installation can have peaks, beyond the agreed values, which may be balanced by valleys.

Being more specific, we may say that, MSU/h is a WLM metric calculated from the total CPU time amount consumed in an LP. This calculation is redone every five minutes in a 4-hour-rolling average (a 4HRA). It means that at every five minutes, a current value replaces the 4-hour average from 5 minutes prior, and a new 4-hour average is recalculated. The software products are charged using the monthly peak hour of the 4HRA.

An installation can implement soft capping (also called *defined capacity*) to enforce a limit on LP consumption, and consequently to save on software fees. For example, if during the month the 4HRA is 200 MSU/h and for just a few hours is 400 MSU/h, the idea is to cap the LP at 200 MSU/h level. It is clear that when the LP is capped, you may face performance problems.

Defined capacity is controlled by WLM and is executed by the PR/SM hypervisor code. As illustrated in the example in Figure 4-14,

- ▶ Curve 1 is the defined capacity, set at 40 MSU/hour.
- ▶ Curve 2 is the hourly 4HRA.
- ▶ Curve 3 is the capped 4HRA.

## 4.18 Group Capacity

LPAR (CEC total capacity 400 MSU/hr)		
LP A (W=465)	LP B (W=335)	LP C (W=200)
Target MSU = 186	Target MSU = 134	Target MSU = 80
No Soft cap	No Soft cap	Soft cap = 30
Maximum = 200	Maximum = 200	Maximum = 30
Protection = 99	Protection = 71	Protection = 30
Capacity Group Prod = 200 MSU/hr		

Figure 4-15 Group Capacity

### Group Capacity

Group Capacity is an extension of Defined Capacity capping, that adds more flexibility. Instead of Defined Capacity where each LP is capped individually, an installation caps a *group* of LPs containing shared logical processors. WLM and PR/SM 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. That is, in the same group, you can have z/OS from different sysplexes, but in the same CPC.

Looking at Figure 4-15, we have a CPCD with three LPs, with the following weights 465, 335, and 200. This CPC has a total capacity of 400 MSU/h according with IBM tables. Then, the target MSU/h and also the guaranteed 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 individually, and C is soft capped at 30 MSU/h. Also, the three belong to a capacity group named Prod, which is capped at 200 MSU/h.

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 software capping (Defined Capacity) 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.

The distribution of MSU/h between LP a and LP B is proportional to the Weigh% of each LP.

## **Intelligent Resource Director (IRD)**

Parallel Sysplex provides 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 functions**

IRD is not actually a product or a system component. Rather, it is three separate but mutually supportive functions:

- ▶ WLM PR/SM CP Management - Workload Manager distributes processor resources across an LP cluster by dynamically adjusting the LPAR weights in response to changing workload requirements.
- ▶ Dynamic Channel Path Management (DCM)
- ▶ Channel Subsystem I/O Priority Queuing (CSS IOPQ) - see "Channel Subsystem I/O Priority Queuing" on page 142.

For various reasons, the two first IRD functions are not often used by customers. So at this "ABCs of z/OS System Programming" introductory level, we only cover the last function, Channel Subsystem I/O Priority Queuing.

## **Channel Subsystem I/O Priority Queuing**

Channel Subsystem I/O Priority Queuing is a capability delivered on z14 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. Sometimes this approach caused high priority work to be delayed behind low-priority work.

With Channel Subsystem Priority Queuing, if an important task is missing its WLM goals due to I/O contention, for example, on storage controllers shared with other tasks, WLM gives a higher channel subsystem I/O priority over the less important ones. Then, Channel Subsystem Priority Queuing ensures that the PU personality named SAP organizes any eventual I/O operations queue by the I/O priority figure coming directly from the WLM. SAP is a PU in charge of guaranteeing that an I/O operation (a dialog between the channel and the controller) really starts. Because this IRD has a CPC scope, you may use that to privilege I/O operations coming from a z/OS production LP in relation to a sandbox z/OS LP.

This IRD recommended functionality must be explicitly ordered from the HMC hardware console.

For more information about this topic, refer to *z/OS Intelligent Resource Director*, SG24-5952.



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redbooks publication.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 146. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IBM Z Connectivity Handbook*, SG24-5444
- ▶ *IBM z14 Technical Introduction*, SG24-8450
- ▶ *IBM z14 Technical Guide*, SG24-8451
- ▶ *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
- ▶ *Installation Manual for Physical Planning*, GC28-6965-00a
- ▶ *z/OS Distributed File Service zSeries File System Administration*, SC24-5989
- ▶ *OSA-Express Customer's Guide and Reference*, SA22-7935
- ▶ *z/OS HCD User's Guide*, SC34-2669
- ▶ *z/OS HCD Planning*, GA32-0907

## Online resources

These websites 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](http://ibm.com/redbooks)





# ABCs of z/OS System Programming Volume 10

SG24-6990-05

ISBN 0738443107

(1.5" spine)  
1.5" <-> 1.998"  
789 <-> 1051 pages



# ABCs of z/OS System Programming Volume 10

SG24-6990-05

ISBN 0738443107

(1.0" spine)  
0.875" <-> 1.498"  
460 <-> 788 pages



**Redbooks**

## ABCs of z/OS System Programming Volume 10

SG24-6990-05

ISBN 0738443107

(0.5" spine)  
0.475" <-> 0.873"  
250 <-> 459 pages



**Redbooks**

## ABCs of z/OS System Programming Volume 10

(0.2" spine)

0.17" <-> 0.473"

90 <-> 249 pages

(0.1" spine)

0.1" <-> 0.169"

53 <-> 89 pages



# ABCs of z/OS System Programming Volume 10

SG24-6990-05

ISBN 0738443107

(2.5" spine)  
2.5" <-> mmm.n"  
1315 <-> mmm pages



# ABCs of z/OS System Programming Volume 10

SG24-6990-05

ISBN 0738443107

(2.0" spine)  
2.0" <-> 2,498"  
1052 <-> 1314 pages







SG24-6990-05

ISBN 0738443107

Printed in U.S.A.

Get connected

