SA23-2643-03

POWERstation and POWERserver

Hardware Technical Information
General Architectures

Fourth Edition (October 1993)

This edition notice applies to the *POWERstation and POWERserver Hardware Technical Information-General Architectures*. This edition obsoletes all previous editions.

Table of Contents

About This Book

This manual describes architecture features that are common to the system family.

Note: The information in this book can also be found in the CD-ROM Hypertext Information Base Library. This online documentation is designed for use with the InfoExplorer hypertext retrieval system.

Who Should Use This Book

This book is an overview of the operation of the system. It is intended for programmers and engineers who understand computer architecture and programming concepts and who develop hardware and software products for the system family.

How to Use This Book

Overview of Contents

This book contains the following chapters:

- Chapter 1, "Processors," describes the central electronics complex, the document conventions, a general systems overview, instruction formats, and memory addressing.
- Chapter 2, "System I/O Structure," describes bit and byte numbering conventions, I/O bus protocols, the programming model, load and store instructions, the translation, protection, and TCW table, the bus master, the DMA slave, IOCC Commands, Buffer Flush commands, I/O interrupts, special facilities, the system I/O and standard I/O, exception reporting and handling, and implementation details.
- Chapter 3, "Vital Product Data," contains the keyword descriptor summary, the hardware VPD descriptor summary, the Micro Channel adapter requirements, and a sample layout of the Micro Channel adapter VPD.
- Chapter 4, "Initial Program Load (IPL) ROM," describes IPL ROM components, IPL ROM functional characteristics, and error codes.

Overview of Reference Library Contents

This general information manual, is one part of the hardware technical information library. This manual describes features that are common to the system family. Since the last edition, new products have evolved that feature economy of cost and size. Check the front of each chapter in this manual for a note specifying which models are covered in the chapter. The *General Architectures* manual should be used in conjunction with the following hardware technical information manuals:

- *POWERstation and POWERserver Hardware Technical Information-Options and Devices* (SA23-2646)
- *7011 POWERstation and POWERserver Hardware Technical Information* (SA23-2666)
- *7012 POWERstation and POWERserver Models 34x, 35x, 36x, and 37x Hardware Technical Information* (SA23-2680)
- *7013 POWERstation and POWERserver Models 550L, 57x, 58x, 58H, and 590 Hardware Technical Information* (SA23-2684)
- *7015 POWERserver Models 97x, 98x, and 99x Hardware Technical Information* (SA23-2686).

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, keywords, files, directories, and other items whose names are predefined by the system.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Related Publications

The following is a list of related publications. For information on ordering these publications, contact your authorized dealer or marketing representative.

- *IBM RISC System/6000 System Overview* (GC23-2406)
- *Personal System/2 Hardware Interface Technical Reference: Architectures* (S84F-9808)
- *AIX Version 3.2 Assembler Language Reference* (SC23-2197)
- *AIX Version 3.2 Kernel Extensions and Device Support Programming Concepts* (SC23-2207)
- *AIX Version 3.2 Problem Solving Guide and Reference* (SC23-2204).

Ordering Publications

You can order IBM publications from your IBM sales representative or, in the U.S., from IBM Customer Publications Support at 1 800 879-2755. If you believe you are entitled to publications that were not shipped with your RISC System/6000 or AIX purchases, contact your IBM sales representative or Customer Publications Support for assistance.

To order additional copies of this book, use Order Number SA23-2643.

Chapter 1. System Processors

Chapter Contents

Description

This section describes the central electronics complex (CEC) for the POWER2 and POWER implementations of the RISC System/6000, the document conventions, a general systems overview, instruction formats, and memory addressing.

A POWER processor is used in this system family. Like earlier processors, the POWER processor employs a simple register-oriented instruction set that is completely hardwired, and features a pipelined implementation and an efficient storage hierarchy. This enables the processor chip set to run an instruction almost every cycle. Unlike earlier processors, however, this unit employs several advanced architectural and implementation features including separate instruction and data caches, zero-cycle branches, multiple instruction dispatch, simultaneous running of fixed- and floating-point operations, and overlapped running of register-register (RR) operations and load and store commands. As such, the unit combines the simplicity of an instruction set with sophisticated hardware design techniques to achieve a short cycle time and a low cycles-per-instruction (CPI) ratio.

In the POWER2 implementation, six instructions can be executed in a single cycle: a branch, two fixed-point, two floating-point, and a Condition register logical instruction. Counting the floating-point multiply-add instruction as two operations, this yields a peak run rate of eight operations per cycle. In the POWER implementation, four instructions can be executed in a single cycle: a branch, a fixed-point, a floating-point, and a Condition register logical instruction. Counting the floating-point multiply-add instruction as two operations, this yields a peak run rate of five operations per cycle.

Note: This chapter provides information for system models 32x, 34x, 35x, 36x, 37x, 52x, 53x, 540, 55x, 56x, 57x, 58x, 58H, 59x, 730, 930, 95x, 97x, 98x, and 99x. Information for other system models can be found in the product-specific technical information manual for those models.

The processor chip sets described in this chapter are representative of the chip sets used in the models mentioned in the preceding paragraph. The megahertz number of the processor chip set varies depending on the system model.

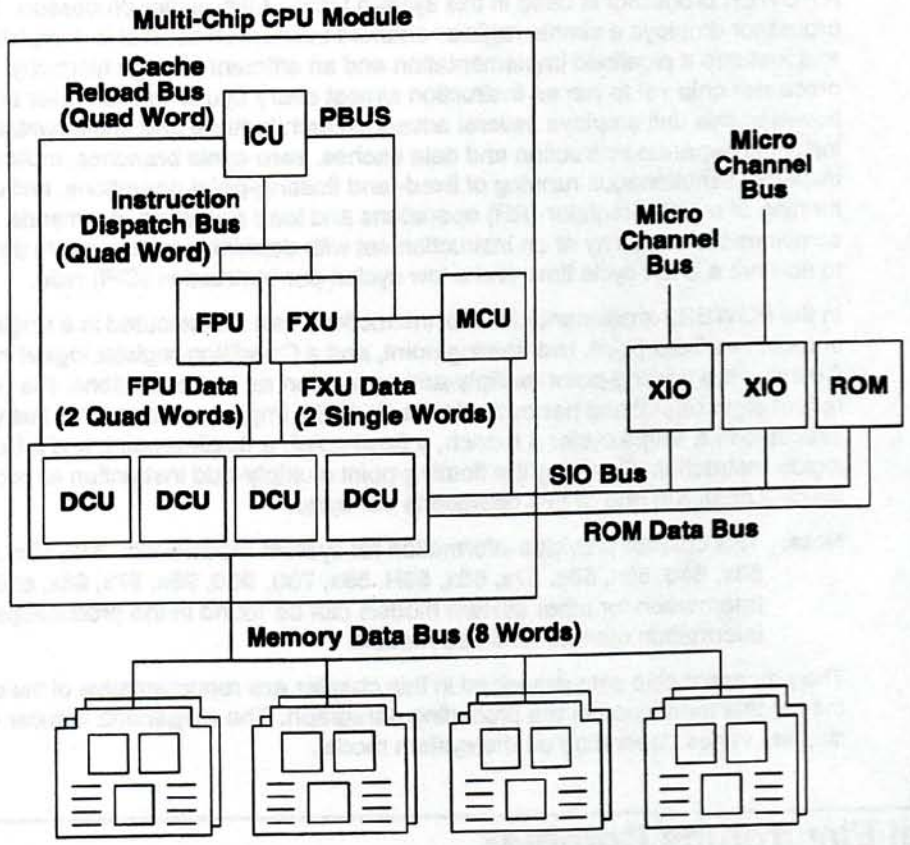
Central Electronics Complex

The POWER and POWER2 processor chip sets form the central electronics complex (CEC) and have up to eleven semi-custom chips: a fixed-point unit (FXU), a floating-point unit (FPU), an instruction cache and branch processing unit (ICU), four data cache units (DCU), a memory control unit (MCU), an Input/Output (I/O) Channel controller a Serial Optical Channel converter, and a clock chip (CLK). Every memory board contains two data multiplexing modules and one control module for interleaving.

There are four basic processor chip sets in this family of system units. The first chip set shown in Figure 1 on page 1-6 is the POWER2 implementation having the following characteristics:

- Fixed-point unit with two execution units
- Floating-point unit with two multiply add units
- 32K-byte instruction cache
- 256K-byte data cache.

This implementation supports configurations with two, four, or eight memory boards. A two memory board configuration supports a 128K-byte data cache and a 128-bit memory interface. A four or eight board configuration supports a 256K-byte data cache and a 256-bit memory interface. The eight chips (ICU, FXU, FPU, 4X DCU, and MCU) are packaged on a multichip carrier. The I/O subsystem can contain up to two extended input/output (XIO) modules.



Note: **Some systems have only two memory boards.**
Some systems have only one XIO module.

POWER2 System Configuration, 8 Word Memory Bus

Figure 1. First Processor Chip Set

The second chip set has two data cache units and a system memory interface that is 64 bits wide.

The third chip set uses the same modules, but has four data cache units and a 128-bit bus to system memory.

The second and third chip sets a have an instruction cache unit with 8K bytes. It has an input and output unit (IOU) that combines the I/O channel controller (for Micro Channel bus) and the serial link logic. Figure 2 shows the chip sets described previously with four data cache units like the second chip set.

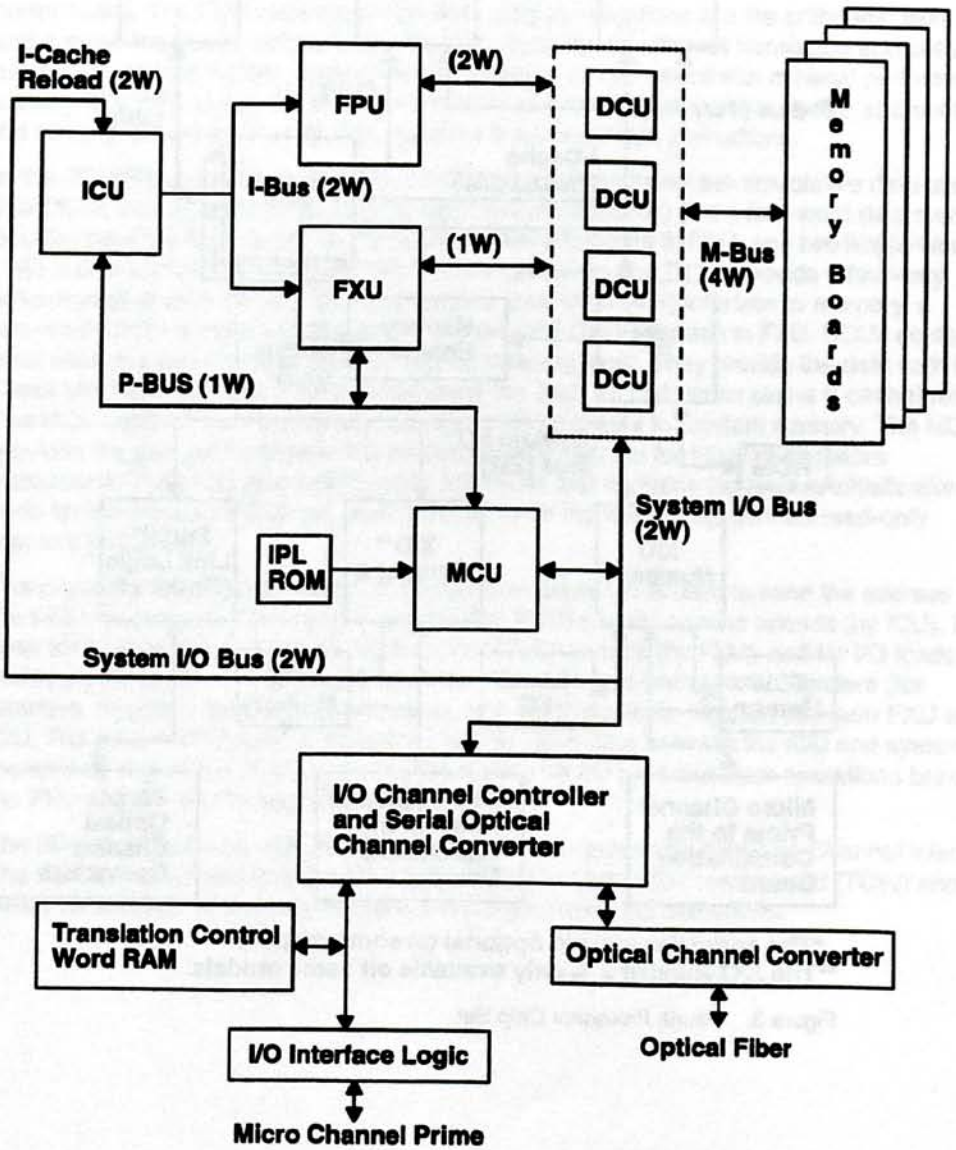
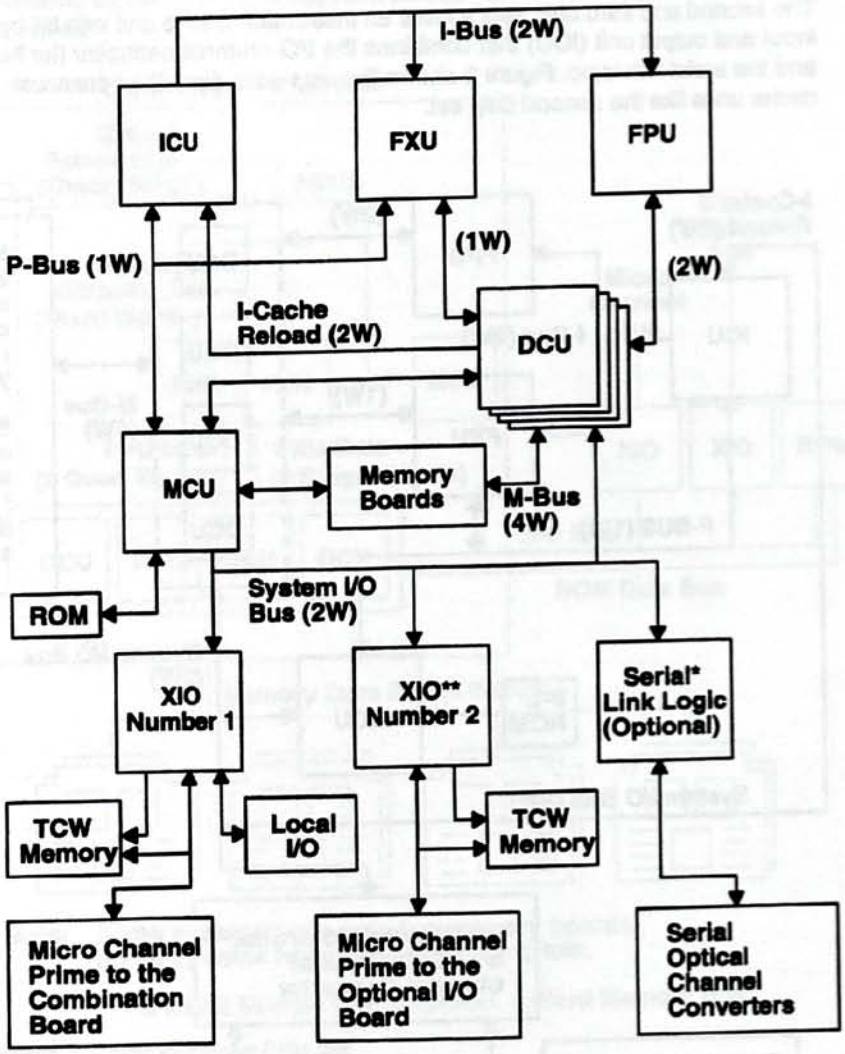


Figure 2. Second and Third Processor Chip Sets

The fourth chip set is shown in Figure 3. This chip set has an instruction cache unit with 32K bytes of chip memory. The I/O subsystem can contain up to two XIO modules and a serial link logic for system serial optical channels. The XIO module, contains the I/O channel control unit that generates the Micro Channel interface.



***The serial link logic is optional on some models.**
****The XIO Number 2 is only available on some models.**

Figure 3. Fourth Processor Chip Set

The POWER2 and POWER implementations have an ICU that contains a two-way set-associative instruction cache. It runs branch instructions and Condition register logical instructions, and supports interrupts. In many cases, branches cost zero cycles because the ICU looks ahead in the instruction stream and removes branches from the stream. In a given cycle, the ICU in the POWER implementation can dispatch two instructions (two to the FXU, or two to the FPU, or one to the FXU and one to the FPU) by way of the I-bus shown in Figure 1 on page 1-6. The ICU in the POWER2 implementation can dispatch four instructions. The floating-point unit contains a full 64-bit double-precision floating-point data flow and conforms to the IEEE 754 binary floating-point standard with software support. Floating-point instructions can run in parallel with fixed-point instructions for maximum performance. The FXU contains the general purpose registers and the arithmetic logic units, and runs all fixed-point instructions. The FXU includes an address translation and data protection unit that makes precise interrupts easier to implement with minimal performance penalty. The FXU also provides the directories and control for the data cache, and controls the running of fixed-point load, floating-point load, and store instructions.

In the POWER2 implementation four DCUs provide a four-way set-associative data cache. The DCUs form an eight-word (four or eight memory boards) and a four-word (two memory boards) interface to memory, two four-word (4W) interfaces to FPU, and two single-word (1W) interfaces to FXU. In the POWER implementation four DCUs provide a four-way set-associative data cache. The DCUs form a four-word (4W) interface to memory, a two-word (2W) interface to FPU, and a single-word (1W) interface to FXU. DCUs contain error checking and correction (ECC) and bit steering logic. They provide the data path for Direct Memory Accesses (DMA), and supply the path for instruction cache (I-cache) reloads. The MCU contains the controls and configuration registers for system memory. The MCU provides the data path between I/O and processor chip set for I/O load and store instructions. The MCU also interfaces to the ROM that contains the system initialization code for the processor chip set, also referred to as the initial program load read-only memory (IPL ROM).

The processor bus (P-bus) shown in Figure 1 on page 1-6 is used to send the address to the MCU for D-cache (data cache) reloads (by FXU) and for I-cache reloads (by ICU). It is used for I-cache translation look-aside buffer (TLB) reloads (by FXU), and for I/O loads and stores (by FXU). The P-bus is also used for moves to and from special registers (for example, Segment registers, Link register, and Machine State register) between FXU and ICU. The system I/O bus is used to transfer the DMA data between the IOU and system memory by way of the DCU, and provides a path for I/O load and store operations between the FXU and the IOU by way of the MCU.

The I/O unit contains an I/O channel control unit that generates the Micro Channel interface. The I/O channel control unit uses the data stored in translation control word (TCW) and tag tables for address translation and data protection during I/O operations.

Document Conventions

The following conventions are used throughout this document:

- Quadwords are 128 bits, doublewords are 64 bits, words are 32 bits, halfwords are 16 bits, bytes are 8 bits.
- All numbers are decimal unless specified in some special way.
- b'*nnn*' means a number expressed in binary format.
- x'*nnn*' means a number expressed in hexadecimal format.
- n x b'0' means *n* zeros.
- n x b'1' means *n* ones.
- (RA|0) means the contents of register RA if the RA field has the value 1 through 31, or the value 0 if the RA field is 0.
- (R*x*) means the contents of register R*x*.
- (FR*x*) means the contents of register FR*x*.
- X(*p*) means bit p of register or field X.
- X sub $_p$ means bit p of register or field X.
- X(*p–q*) means bits p through q of register or field X.
- X(*p..q*) means bits p through q of register or field X.
- X sub $_{p-q}$ means bits p through q of register or field X.
- ¬(RA) means the ones complement of the contents of register RA.
- /, //, ///, ... means a field that is ignored by the hardware.
- The symbol || is used to describe two fields that are appended or concatenated to each other. For example, 010||111 is the same as 010111.
- All bits in registers that are reserved are 0 on read and can be either 0 or 1 on write.
- 2**n means 2 raised to the $n^{**\,th}$ power.
- Field i refers to bits 4 x i to (4 x i) + 3 of a register.
- Positive means greater than 0.
- Negative means less than 0.
- Instructions are assumed to be nonprivileged unless stated otherwise in the instruction description.

Systems Overview

The processor or processor unit contains the sequencing and processing controls for instruction fetch, instruction execution, and interrupt action. The following classes of instructions can be executed by the processing unit:

- Branch processor instructions
- Fixed-point processor instructions
- Floating-point processor instructions.

Refer to *AIX Version 3.2 Assembler Language Reference* for information on a specific instruction.

See Figure 4 for a representation of the logical partitioning provided by the system architecture. The processing unit is a word-oriented fixed-point processor and in a doubleword-oriented floating-point processor. The system architecture uses 32-bit word-aligned instructions and provides for byte, halfword, word, and doubleword operand fetches and stores between system memory and a set of 32 general purpose registers (GPRs), and between system memory and a set of 32 floating-point registers (FPRs).

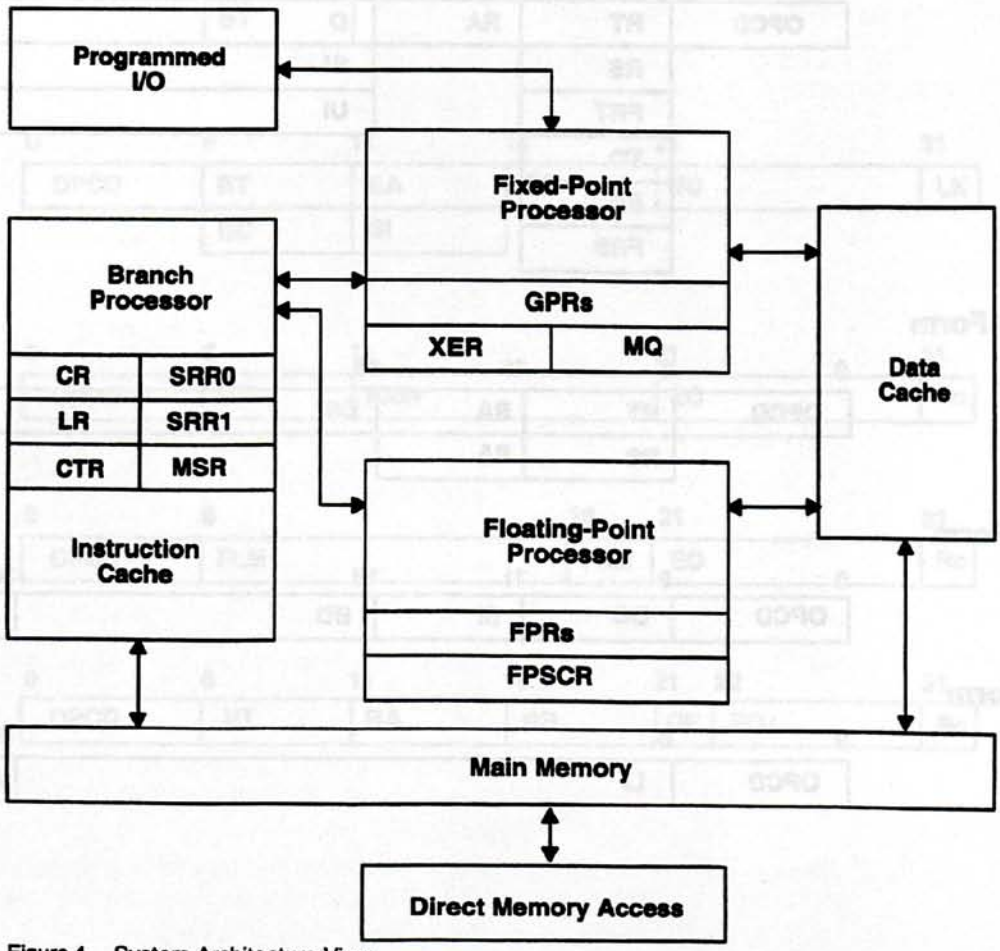


Figure 4. System Architecture View

Instruction Formats

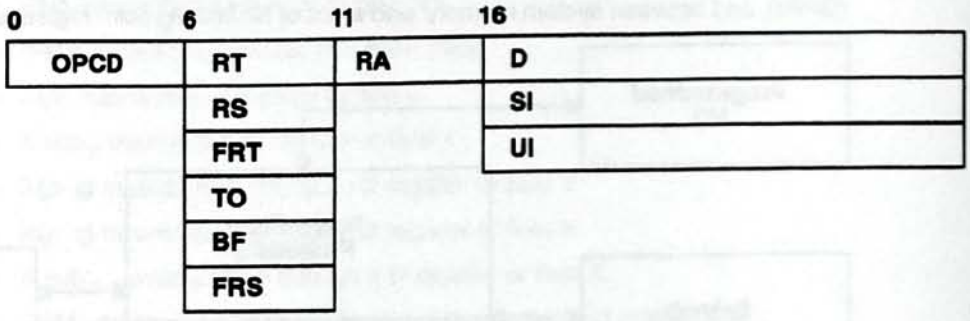
All instructions are 4 bytes long and are located on word boundaries. Thus, whenever instruction addresses are presented to the processing unit (as in branch instructions) the two low-order bits are ignored. Similarly, whenever the processing unit develops an instruction address, its two low-order bits are 0.

Bits 0 through 5 always specify the opcode. For XO-form instructions, an extended opcode is specified in bits 22 through 30. For all other X-form instructions, an extended opcode is specified in bits 21 through 30. For A-form instructions, an extended opcode is specified in bits 26 through 30.

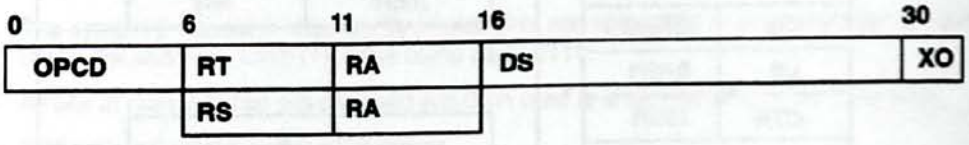
The remaining bits contain one or more alternative fields for the different instruction formats.

Forms

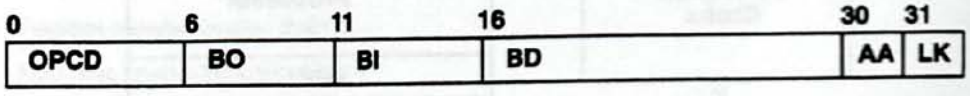
D Form



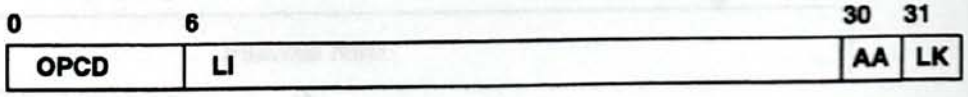
DS Form



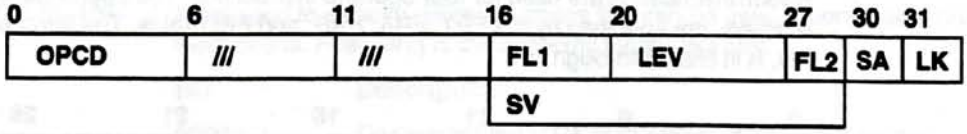
B Form



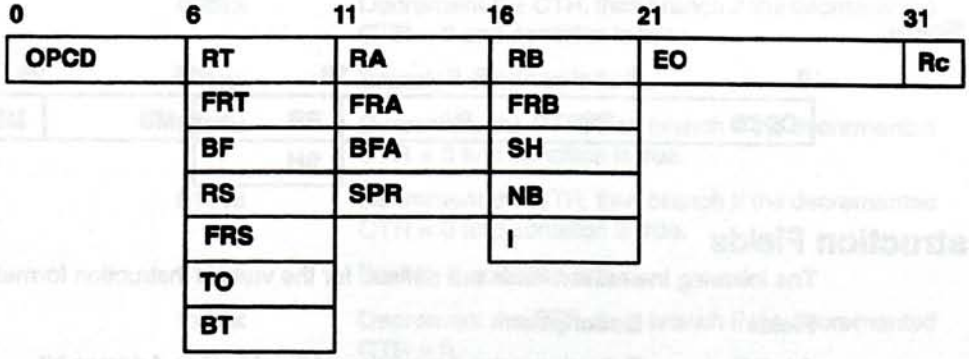
I Form



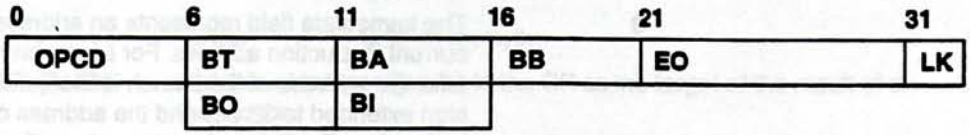
SC Form



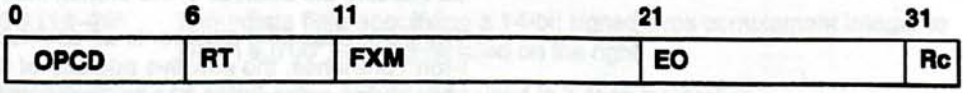
X Form



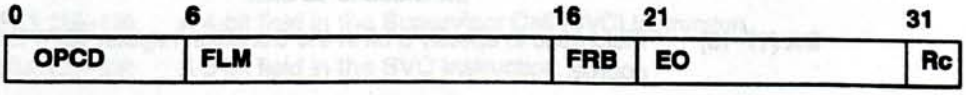
XL Form



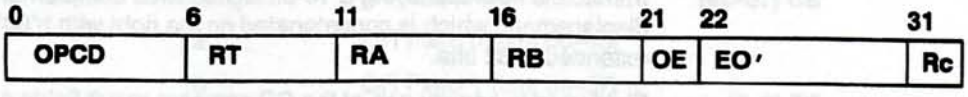
XFX Form



XFL Form

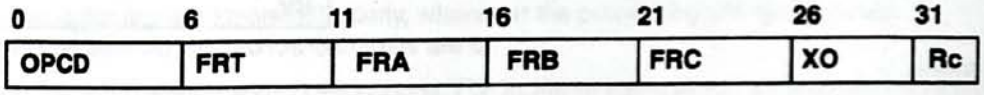


XO Form

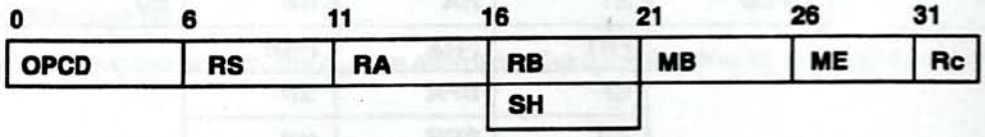


A Form

A-form instructions are used for four operand instructions. The operands, all floating-point registers, are specified by the FRT, FRA, FRB, and FRC fields. The short extended opcode, XO, is in bits 26 through 30.



M Form



Instruction Fields

The following instruction fields are defined for the various instruction formats:

Fields	Description
AA (30)	Following is the description of the Absolute Address bit.

Bit	Description
0	The immediate field represents an address relative to the current instruction address. For I-form branches, the effective address of the branch is the sum of the LI field sign extended to 32 bits and the address of the branch instruction. For B-form branches, the effective address of the branch is the sum of the BD field sign extended to 32 bits and the address of the branch instruction.
1	The immediate field represents an absolute address. For I-form branches, the effective address of the branch is the LI field sign extended to 32 bits. For B-form branches, the effective address of the branch is the BD field sign extended to 32 bits.

BA (11–15)	Field used to specify a bit in the Condition register (CR) to be used as a source.
BB (16–20)	Field used to specify a bit in the CR to be used as a source.
BD (16–29)	Immediate field specifying a 14-bit signed twos complement branch displacement, which is concatenated on the right with b'00' and sign extended to 32 bits.
BF (6–8)	Field used to specify one of the CR compare result fields or one of the FPSCR fields as a target. If i = BF(6–8), then field i refers to bits i x 4 to (i x 4) + 3 of the register.
BFA (11–13)	Field used to specify one of the CR compare result fields, one of the FPSCR fields, or one of the XER fields as a source. If j = BFA(11–13), then field j refers to bits j x 4 to (j x 4) + 3 of the register.
BI (11–15)	Field used to specify the bit in the CR to be used as the condition of the branch.

Fields	Description
BO (6–10)	Field used to specify different options that can be used in conditional branch instructions. Following is the encoding for the BO field:

BO	Description
0000x	Decrement the CTR, then branch if the decremented CTR ≠ 0 and condition is false.
0001x	Decrement the CTR, then branch if the decremented CTR = 0 and condition is false.
001xx	Branch if condition is false.
0100x	Decrement the CTR, then branch if the decremented CTR ≠ 0 and condition is true.
0101x	Decrement the CTR, then branch if the decremented CTR = 0 and condition is true.
011xx	Branch if condition is true.
1x00x	Decrement the CTR, then branch if the decremented CTR ≠ 0.
1x01x	Decrement the CTR, then branch if the decremented CTR = 0.
1x1xx	Branch always.

BT (6–10)	Field used to specify a bit in the CR as the target of the result of an instruction.
D (16–31)	Immediate field specifying a 16-bit signed twos complement integer sign extended to 32 bits.
DS (16–29)	Immediate field specifying a 14-bit signed twos complement integer to which a b'00' is concatenated on the right.
EO (21–30)	A 10-bit extended opcode used in X-form instructions.
EO' (22–30)	A 9-bit extended opcode used in XO-form instructions.
FL1 (16–19)	A 4-bit field in the Supervisor Call (SVC) instruction.
FL2 (27–29)	A 3-bit field in the SVC instruction.
FXM (12–19)	Field mask, identifies which CR field is to be updated.

Bit	Description
12	CR Field 0 (bits 00–03)
13	CR Field 1 (bits 04–07)
14	CR Field 2 (bits 08–11)
15	CR Field 3 (bits 12–15)
16	CR Field 4 (bits 16–19)
17	CR Field 5 (bits 20–23)
18	CR Field 6 (bits 24–27)
19	CR Field 7 (bits 28–31).

Fields	Description
FLM (7–14)	Field mask, identifies which FPSCR field is to be updated.

Bit	Description
7	FPSCR Field 0 (bits 00–03)
8	FPSCR Field 1 (bits 04–07)
9	FPSCR Field 2 (bits 08–11)
10	FPSCR Field 3 (bits 12–15)
11	FPSCR Field 4 (bits 16–19)
12	FPSCR Field 5 (bits 20–23)
13	FPSCR Field 6 (bits 24–27)
14	FPSCR Field 7 (bits 28–31).

FRA (11–15)	Field used to specify an FPR as a source of an operation.
FRB (16–20)	Field used to specify an FPR as a source of an operation.
FRC (21–25)	Field used to specify an FPR as a source of an operation.
FRS (6–10)	Field used to specify an FPR as a source of an operation.
FRT (6–10)	Field used to specify an FPR as the target of an operation.
I (16–19)	Immediate field used as the data to be placed into a field in the FPSCR.
LEV (20–26)	Immediate field in the SVC instruction that addresses the SVC routine by b'1' || LEV || b'00000' if SA = 0.
LI (6–29	Immediate field specifying a 24-bit signed two's complement integer that is concatenated on the right with b'00' and sign extended to 32 bits.
LK (31)	Following is the description of the Link bit.

Bit	Description
0	Do not set the Link register.
1	Set the Link register. If the instruction is a branch, the address of the instruction following the branch instruction is placed into the Link register. If the instruction is an SVC, the address of the instruction following the SVC instruction is placed into the Link register.

Fields	Description
MB (21–25) & ME (26–30)	Fields used to specify a 32-bit string, consisting of either a substring of ones surrounded by zeros or a substring of zeros surrounded by ones. The encoding is as follows:

Fields	Description
MB (21–25)	Index to start bit of substring of ones.
ME (26–30)	Index to stop bit of substring of ones.

Let *mstart* = MB and *mstop* = ME.

If *mstart* < *mstop* + 1
then mask (*mstart*..*mstop*) = ones
mask (all other) = zeroes.

If *mstart* = *mstop* + 1 then
mask (0–31) = ones.

If *mstart* > *mstop* + 1 then
mask (*mstop* + 1..*mstart* –1) = zeros
mask (all other) = ones.

NB (16–20)	Field used to specify the number of bytes to move in a load or store string immediate.
OPCD (0–5)	The basic opcode field of the instruction.
OE (21)	Used for extended arithmetic to inhibit the setting of OV and SO in XER.
RA (11–15)	Field used to specify a GPR to be used as a source or as a target.
RB (16–20)	Field used to specify a GPR to be used as a source.
Rc (31)	Following is the description of the Record bit.

Setting	Description
0	Do not set the Condition register (CR).
1	Set the Condition register to reflect the result of the operation.

For fixed-point instructions, CR bits (0 to 3) are set to reflect the result as a signed quantity. The result as an unsigned quantity or a bit string can be deduced from the EQ bit.

For floating-point instructions, CR bits (4 to 7) are set to reflect Floating-Point Exception, Floating-Point Enabled Exception, Floating-Point Invalid Operation Exception, and Floating-Point Overflow Exception.

RS (6–10)	Field used to specify a GPR to be used as a source.
RT (6–10)	Field used to specify a GPR to be used as a target.

Fields	Description
SA (30)	The following describes the SVC Absolute.

Setting	Description
0	SVC routine at address '1' || LEV || b'00000'.
1	SVC routine at address X'1FE0'.

SH (16–20)	Field used to specify a shift amount.
SI (16–31)	Immediate field used to specify a 16-bit signed integer.
SPR (11–15)	Special Purpose register.

SPR	Special Purpose Register
00000 (00)	MQ
00001 (01)	XER
00100 (04)	from RTCU
00101 (05)	from RTCL
00110 (06)	from DEC
01000 (08)	LR
01001 (09)	CTR
10100 (20)	to RTCU
10101 (21)	to RTCL
10110 (22)	to DEC
11010 (26)	SRR 0
11011 (27)	SRR 1.

TO (6–10)	TO bit ANDed with condition.

TO bit	ANDed with Condition
6	Compares less than.
7	Compares greater than.
8	Compares equal.
9	Compares logically less than.
10	Compares logically greater than.

UI (16–31)	Immediate field used to specify a 16-bit unsigned integer.
XO (26–30)	A-form instructions contain a 5-bit extended opcode.
XO (30, 31)	DS-form instructions contain a 2-bit extended opcode.

Memory Addressing

Within the context of a program executing on the processing unit (PU), system memory is organized into doublewords, words, halfwords, and bytes, which are constrained to lie on boundaries that are multiples of their sizes. See Figure 5 for an example of byte, halfword, word, doubleword, and quadword memory addressing.

Bits	Addresses				
	Byte	Halfword	Word	Doubleword	Quadword
0–31	0000	0000	0000	0000	0000
	0001				
	0010	0010			
	0011				
31–63	0100	0100	0100		
	0101				
	0110	0110			
	0111				
32–63	1000	1000	1000	1000	
	1001				
	1010	1010			
	1011				
64–127	1100	1100	1100		
	1101				
	1110	1110			
	1111				

Figure 5. Memory Organization

Bytes in system memory are consecutively numbered starting with 0. Each number is the address of the corresponding byte. The 32-bit addresses computed for system memory access are termed *effective addresses* and specify a byte in memory. System memory address arithmetic wraps around from the maximum byte address, $2^{32} - 1$, to address 0.

System memory can be accessed by quadword, doubleword, word, halfword, or byte. The required number of bytes are fetched from a properly aligned area of memory. The rules when the operands are not properly aligned are controlled by a mode bit, MSR(AL). See "Machine State Register" on page 1-22.

The mapping to *real memory* addresses is controlled by relocate (address translation) facilities. When the relocate facility is active, effective addresses generated by program execution are first transformed to 52-bit *virtual address*, which in turn are mapped to real memory.

In general, the terms *memory* and *address* are used within the context of the effective addresses generated by the PU.

All processor computations are performed in registers in the processing unit (PU). There are no instructions, for instance, to add two numbers, one of which is in memory.

Effective Address Calculation

Effective addresses (EAs) are generated by instructions that reference data in system memory and by taken branch instructions. Address calculations use 32-bit two's complement binary arithmetic. A carry from bit 0 is ignored.

A value of 0 in the RA field indicates the absence of the corresponding address component. For the absent component, a 0 value is used in forming the address. This is shown in the instruction descriptions as (RA|0).

X-form instructions are used for data references. Address computation adds the GPR contents designated by the RA field or the value 0 if RA equals a value of 0 with the GPR contents designated by the RB field. The computation is shown as (RA|0) + (RB).

With D-form instructions, the 16-bit D field is sign extended to form a 32-bit address component. In computing the effective address of a data element, this address component is added to the GPR contents designated by the RA field or the value 0 if RA equals a value of 0.

With DS-form instructions, the 2-bits of zeros are added to the 14-bit DS field which is then sign extended to form a 32-bit address component. In computing the effective address of a data element, this address component is added to the GPR contents designated by the RA field or the value 0 if RA equals a value of 0.

With I-form branch instructions, the 24-bit LI field is concatenated on the right with b'00' and sign extended to form a 32-bit address. When AA equals a value of 0, this address is added to the address of the branch instruction to form the effective address. If AA equals a value of 1, this 32-bit value is the effective address.

With B-form branch instructions, the 14-bit BD field is concatenated on the right with b'00' and sign extended to form a 32-bit value. If AA equals a value of 0, this 32-bit value is added to the address of the branch instruction to form the effective address. If AA equals a value of 1, this 32-bit value is the effective address.

With XL-form branch instructions, bits 0 to 29 of the Link register or the Count register are concatenated on the right with b'00' to form the effective address.

Branch Processor

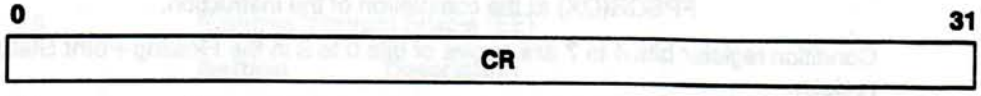
This section describes the registers and instructions that make up the branch processor facilities.

Branch Processor Registers

This section describes the branch processor registers and their bit definitions.

Condition Register

The Condition register (CR) is a 32-bit register that reflects the result of certain operations and provides a mechanism for testing (and branching).



Bits	Name
00–03	CR Field 0
04–07	CR Field 1
08–11	CR Field 2
12–15	CR Field 3
16–19	CR Field 4
20–23	CR Field 5
24–27	CR Field 6
28–31	CR Field 7.

The Condition register bits are grouped into eight 4-bit fields, named CR Field 0 through CR Field 7, which are set in one of the following ways:

- A load or copy operation into a specific CR field.
- CR Field 0 can be set as the implicit result of a fixed-point operation.
- CR Field 1 can be set as the implicit result of a floating-point operation.
- As the result of either a fixed or floating-point compare operation into a specified CR field.

Instructions are provided to test these bits singly and in combination.

When the record bit (Rc) is set to 1 in most fixed-point instructions, the first three bits of CR Field 0 are set by a comparison of the result, which is interpreted as a signed integer, to a value of 0. The fourth bit of CR Field 0 is copied from the SO field of the XER. Add Immediate, Add Immediate Lower, and Add Immediate Upper instructions set these four bits implicitly. These bits are interpreted as shown in the following list:

Bit	Description
0	Compares Less Than, Negative (LT). For arithmetic operations, the result is negative or less than a value of 0. For compare operations, (RA) < SI, UI, or (RB).
1	Compares Greater Than, Positive (RB). For arithmetic operations, the result is positive or greater than a value of 0. For compare operations, (RA) > SI, UI, or (RB).
2	Compares Equal, Zero (EQ). For arithmetic operations, the result is a value of 0 or equal to a value of 0. For compare operations, (RA) = SI, UI, or (RB).
3	Summary Overflow (SO). This is a copy of the final state of XER(SO) at the completion of the instruction.

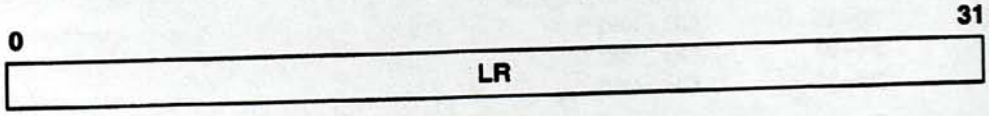
When the Rc bit equals a value of 1 in all floating-point instructions except the Floating-Point Compare instruction, CR Field 1 (Condition register bits 4 to 7) is set to the floating-point exceptions status. These bits are interpreted as shown in the following list:

Bit	Description
4	Floating-Point Exception (FX). This is a copy of the final state of FPSCR(FX) at the completion of the instruction.
5	Floating-Point Enable Exception (FEX). This is a copy of the final state of FPSCR(FEX) at the completion of the instruction.
6	Floating-Point Invalid Operation Exception (VX). This is a copy of the final state of FPSCR(VX) at the completion of the instruction.
7	Floating-Point Overflow Exception (OX). This is a copy of the final state of FPSCR(OX) at the completion of the instruction.

Condition register bits 4 to 7 are copies of bits 0 to 3 in the Floating-Point Status and Control register.

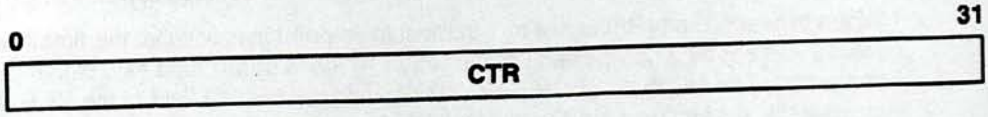
Link Register

The Link register (LR) is a 32-bit register. The Link register provides the branch target address for the Branch Conditional Register instruction and holds the return address (link address) for branch and link type instructions and SVC instructions.



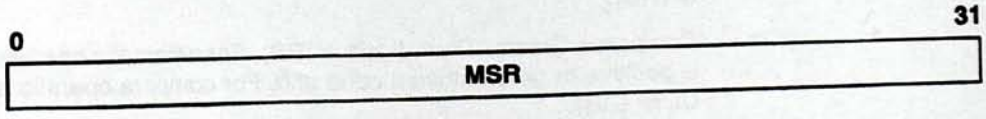
Count Register

The Count register (CTR) is a 32-bit register. The Count register contains a loop count and is automatically decremented during execution of the branch and count instructions, wrapping from X'00000000' around through X'FFFFFFFF'. The Count register also provides the branch target address for the Branch to Count Register instruction. The Count register contains a copy of bits 16 to 31 of MSR and bits 16 to 31 of the SVC instruction after execution of that SVC instruction. Both registers can be copied to and from any GPR.



Machine State Register

The Machine State register (MSR) is a 32-bit register that defines the modal state of the processor. When the RFI instruction is executed, bits 16 to 31 of SRR 1 are placed into bits 16 to 31 of the MSR. The MSR can also be modified by the Move to Machine State Register instruction.



Bit	Name	Description
00–15		Reserved
16	EE	External Interrupt Enable
17	PR	Problem State
18	FP	FP Available
19	ME	Machine Check Enable
20	FE	FP Exception Enable

21	SE	Single-Step Enable
22	BE	Branch and Trap Enable
23	FE	FP Imprecise Enable
24	AL	Alignment Check
25	IP	Interrupt Prefix
26	IR	Instruction Relocate
27	DR	Data Relocate
28		Reserved
29	PM	Performance Monitor Control
30–31		Reserved.

The following are the Machine State register bit definitions and settings:

Bits	Description
0–15	Reserved
16	External Interrupt Enable (EE)

Setting	Description
0	The processor is disabled against external interrupts.
1	The processor is enabled to take external interrupts.

17 Problem State (PR)

Setting	Description
0	The processor is privileged to execute any instruction.
1	The processor can only execute the nonprivileged instructions.

18 Floating-Point (FP) Available

Setting	Description
0	The processor cannot execute any floating-point instructions, including floating-point loads, stores and moves.
1	The processor can execute floating-point instructions.

19 Machine Check Enable (ME)

Setting	Description
0	Machine check interrupts are disabled.
1	Machine check interrupts are enabled.

20 Floating-Point Exception Interrupt Enable (FE)

Setting	Description
0	Program interrupts on floating-point enabled exception are disabled.
1	Program interrupts on floating-point enabled exception are enabled.

Bits	Description
21	Single-Step Enable (SE)

Setting	Description
0	The processor executes instructions normally.
1	The processor generates a Single-Step type Trace Interrupt upon the successful execution of an instruction (the instruction does not cause any other type of interrupt).

22 Branch and Trap Enable (BE)

Setting	Description
0	The processor executes branch instructions normally.
1	The processor generates a Branch and Trap type Trace Interrupt after completing the execution of a branch instruction.

23 FP Imprecise Enable (FE)

Setting	Description
0	FP Imprecise interrupts are disabled.
1	FP Imprecise interrupts are enabled if MSR(FE) = 0.

24 Alignment Check (AL)

Setting	Description
0	Alignment checking is off and the low-order bits of the address are ignored.
1	Alignment checking is on; alignment checking proceeds as follows: If bits 29, 30, or 31 of an address generated by a doubleword data memory reference instruction are nonzero, an alignment interrupt is generated if the hardware cannot perform the unaligned access. If bits 30 or 31 of an address generated by a word data memory reference instruction are nonzero, an alignment interrupt is generated if the hardware cannot perform the unaligned access. If bit 31 of an address generated by a halfword data memory reference instruction is nonzero, an alignment interrupt is generated if the hardware cannot perform the unaligned access. This checking does not apply to the load and store string-type instructions since these instructions always perform the unaligned access. Load and store multiple-type instructions always generate an alignment interrupt if bits 30 to 31 of the effective address are nonzero. When the memory reference is to an I/O segment, the address is sent to I/O unmodified, regardless of the setting of the MSR(AL).

Bits	Description	
25	Interrupt Prefix (IP)	
	Setting	**Description**
	0	Interrupts vectored to the effective address X'000*xxxxx*' where *xxxxx* is the interrupt offset.
	1	Interrupts vectored to the effective address X'FFF*xxxxx*' where *xxxxx* is the interrupt offset. This is intended to direct the interrupt to read only memory (ROM).
26	Instruction Relocate (IR)	
	Setting	**Description**
	0	Instruction address translation is off.
	1	Instruction address translation is on.
27	Data Relocate (DR)	
	0	Data address translation is off.
	1	Data address translation is on.
28	Reserved	
29	Controls performance monitoring functions.	
30–31	Reserved.	

Fixed-Point Processor Registers

This section describes the registers in the fixed-point processor facility.

General Purpose Registers

All manipulation of information is done in registers internal to the processing unit (PU).The principal storage within the fixed-point processor is a set of 32 general purpose registers (GPRs). Each GPR consists of 32 bits. See Figure 6 for an example of the general purpose registers.

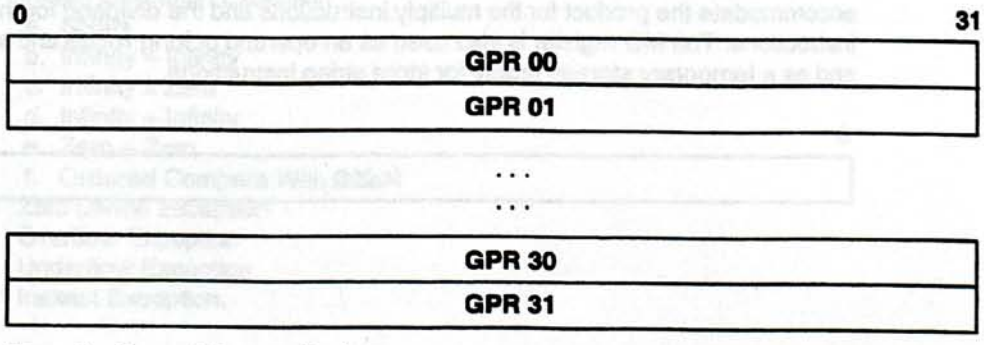
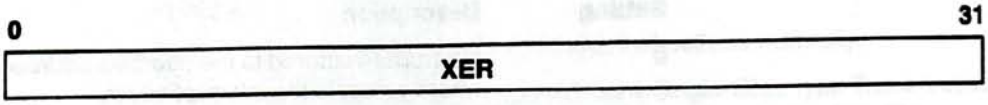


Figure 6. General Purpose Registers

Fixed-Point Exception Register

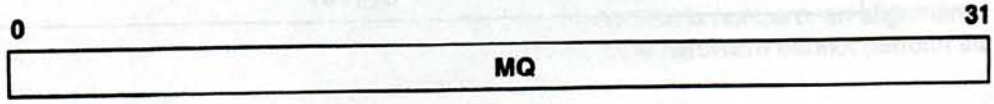
The Fixed-Point Exception register (XER) is in the fixed-point unit and is 32 bits wide.



Bit	Description
0	Summary Overflow (SO) The Summary Overflow bit is set to 1 whenever an instruction sets the Overflow bit to indicate overflow and remains set until software resets it. The SO bit is not altered by the compare instructions.
1	Overflow (OV) The Overflow bit is set to indicate that an overflow has occurred during an instruction operation. In the case of add and subtract instructions, it is set to 1 if the carry out of bit 0 is not equal to the carry out of bit 1. Otherwise the OV bit is set to 0. The OV bit is not altered by the compare instructions.
2	Carry (CA) The Carry bit is set to indicate a carry from bit 0 of the computed result. In the case of add and subtract instructions, it is set to 1 if the operation generates a carry out of bit 0. Otherwise, the CA bit is set to 0. The CA bit is not altered by the compare instructions.
3-15	Reserved
16-23	Used by the Load String and Compare Byte Indexed instructions as the byte being compared against.
24	Reserved
25-31	Used by Load String Indexed, Load String and Compare Byte Indexed, and Store String Indexed instructions to indicate the number of bytes loaded or stored.

Multiply Quotient Register

The Multiply Quotient (MQ) register is a 32-bit register that provides a register extension to accommodate the product for the multiply instructions and the dividend for the divide instructions. The MQ register is also used as an operand of long rotate and shift instructions and as a temporary storage facility for store string instructions.



Floating-Point Processor Overview

The floating-point processor (FPP) provides high-performance execution of floating-point operations. Instructions are provided to perform arithmetic operations in floating-point registers and move floating-point data between memory and these registers.

This architecture provides for hardware to implement a floating-point system as defined in ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating Point Arithmetic*, but is dependent on supporting software to be in conformance with that standard.

A floating-point number consists of a signed exponent and a signed significand. The quantity expressed by this number is the product of the significand and the number 2**exponent. Encodings are provided in the data format to represent finite numeric values, ± Infinity and Not-a-Number (NaN) values. Operations involving infinities produce results obeying traditional mathematical conventions. NaN values have no mathematical interpretation. Their encoding permits a variable diagnostic-information field. They can indicate such things as uninitialized variables and can be produced by certain invalid operations.

There are two classes of exceptional events that occur during instruction execution that are unique to the FPP:

- FPP unavailable
- Floating-point exception.

The FPP unavailable event is signaled with a Floating-Point Not Available Interrupt. Floating-point exceptions are signaled with bits set in the Floating-Point Status and Control register and can generate a precise interrupt with the proper bits enabled.

The Floating-Point Available bit is defined to enhance context switching performance for programs that do not require the use of FPP. The Floating-Point Available bit is defined in "Machine State Register," on page 1-22.

If the Machine State Register (Floating-Point) (MSR(FP)) bit equals 1, the FPP is available for use and floating-point instructions can be successfully executed. If the MSR(FP) bit equals 0, the FPP is unavailable for use, execution of any floating-point instruction is suppressed, and a Floating-Point Unavailable Interrupt is generated to signal the attempted use of the FPP in the unavailable state.

The following floating-point exceptions are detected by the hardware:

- Invalid Operation Exception
  a. SNaN
  b. Infinity – Infinity
  c. Infinity x Zero
  d. Infinity ÷ Infinity
  e. Zero ÷ Zero
  f. Ordered Compare With a NaN
- Zero Divide Exception
- Overflow Exception
- Underflow Exception
- Inexact Exception.

Each floating-point exception and exception sub-class (in the case of Invalid Operation Exception) has an Exception bit defined in the Floating-Point Status and Control Register. Each floating-point exception has an Enable bit defined in the Floating-Point Status and Control Register. See "Floating-Point Status and Control Register" on page 1-29 for definitions of these bits. A bit is defined in the MSR, Floating-Point Exception Interrupt Enable, or MSR(FE), which allows a precise program interrupt to be generated when an enabled floating-point exception occurs.

Floating-Point Registers

Implementations of this architecture provide 32 floating-point registers (FPR). The floating-point instruction formats provide a 5-bit field for specifying the FPRs used in the instruction execution. The FPRs are numbered 0 to 31. See Figure 7 for a representation of the floating-point registers. A Floating-Point Status and Control register controls the handling of floating-point exceptions and records status resulting from the floating-point operations.

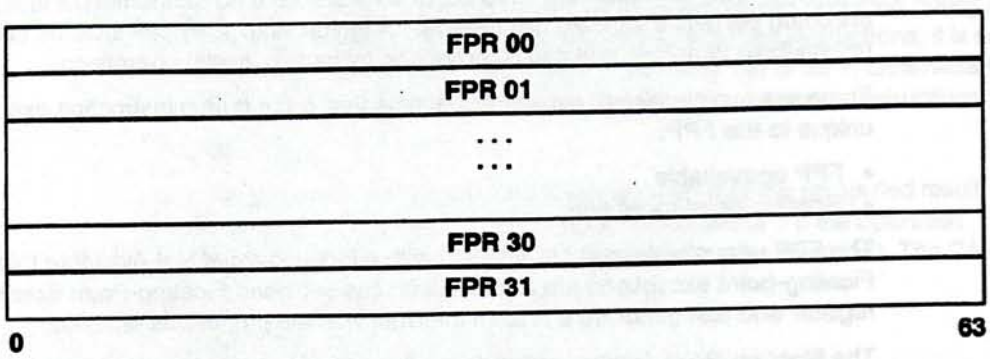


Figure 7. Floating-Point Registers

Each FPR contains 64 bits, which support the double-precision floating-point format. All operations that interpret the contents of an FPR as a floating-point value use the double-precision floating-point format for this interpretation.

All floating-point operations other than load and store operations are performed on operands located in FPRs and place the result value in an FPR. Status information is placed in the Floating-Point Status and Control register and in some cases in the Condition register.

Load and store double instructions are provided that transfer 64 bits of data between memory and the FPRs in the FPP with no conversion. Load single instructions are provided to transfer and convert floating-point values in single floating format from memory to the same value in double floating format in the FPRs. Store single instructions are provided to transfer and convert floating-point values in double floating format from the FPRs to the same value in single-floating format in memory.

Floating-Point Status and Control Register

The Floating-Point Status and Control register (FPSCR) contains the status and control flags for floating-point operations. Bits 0 to 19 are Status bits. Bits 20 to 31 are Control bits.

Bit	Name	Description
00	FX	Floating-Point Exception Summary
01	FEX	Floating-Point Enabled Exception Summary
02	VX	Floating-Point Invalid Operation Exception Summary
03	OX	Floating-Point Overflow Exception
04	UX	Floating-Point Underflow Exception
05	ZX	Floating-Point Zero Divide Exception
06	XX	Floating-Point Inexact Exception
07	VXSNAN	Floating-Point Invalid Operation Exception (SNaN)
08	VXISI	Floating-Point Invalid Operation Exception INF – INF)
09	VXIDI	Floating-Point Invalid Operation Exception (INF + INF)
10	VXZDZ	Floating-Point Invalid Operation Exception (0 + 0)
11	VXIMZ	Floating-Point Invalid Operation Exception (INF x 0)
12	VXVC	Floating-Point Invalid Operation Exception (Invalid Compare)
13	FR	Floating-Point Fraction Rounded
14	FI	Floating-Point Fraction Inexact
15	C	Floating-Point Result Class Descriptor
16	FL	Floating-Point Less Than
17	FG	Floating-Point Greater Than
18	FE	Floating-Point Equal
19	FU	Floating-Point Unordered
20		Reserved
21		Reserved
22	VXSQRT	Floating-Point Invalid Operation Exception (Invalid Square Root)
23	VXCVI	Floating-Point Invalid Operation Exception (Invalid Integer Convert)
24	VE	Floating-Point Invalid Operation Exception Enable
25	OE	Floating-Point Overflow Exception Enable
26	UE	Floating-Point Underflow Exception Enable
27	ZE	Floating-Point Zero Divide Exception Enable
28	XE	Floating-Point Inexact Exception Enable
29		Reserved
30	RN	Floating-Point Rounding Control
31	RN	Floating-Point Rounding Control.

The format of the FPSCR follows:

Bit	Description
0	Floating-Point Exception Summary (FX). Every floating-point arithmetic instruction, floating-point compare instruction, and the Floating Round to Single instruction shall implicitly set FPSCR(FX) if that instruction causes any of the Floating-Point Exception bits in the FPSCR to transition from 0 to 1. Also, use of the mtfsb1 instruction, which causes any of the Floating-Point Exception bits in the FPSCR to transition from 0 to 1 shall implicitly set FPSCR(FX). The mcrfs instruction shall be able to implicitly reset the FPSCR(FX). And finally, the mtfsf, mtfsfi, mtfsb1, and mtfsb0 instructions are able to set or clear FPSCR(FX) explicitly.
1	Floating-Point Enabled Exception Summary (FEX). This bit signals the occurrence of any of the enabled exception conditions. It is the 'OR' of all the floating-point exceptions masked with their respective enable.
2	Floating-Point Invalid Operation Exception Summary (VX). This bit signals the occurrence of any invalid operation exceptions. It is the 'OR' of all the invalid operation exceptions.
3	Floating-Point Overflow Exception (OX). See "Overflow Exception" on page 1-42 for information about this register.
4	Floating-Point Underflow Exception (UX). See "Underflow Exception" on page 1-44 for information about this register.
5	Floating-Point Zero Divide Exception (ZX). See "Zero Divide Exception" on page 1-41 for information about this register.
6	Floating-Point Inexact Exception (XX). See "Inexact Exception"on page 1-44 for information about this register.
7	Floating-Point Invalid Operation Exception (SNaN) (VXSNAN). See "Invalid Operation Exception" on page 1-40 for information about this register.
8	Floating-Point Invalid Operation Exception (INF – INF) (VXISI). See "Invalid Operation Exception" on page 1-40 for information about this register.
9	Floating-Point Invalid Operation Exception (INF + INF) (VXIDI). See "Invalid Operation Exception" on page 1-40 for information about this register.
10	Floating-Point Invalid Operation Exception (0 + 0) (VXZDZ). See "Invalid Operation Exception" on page 1-40 for information about this register.
11	Floating-Point Invalid Operation Exception (INF x 0) (VXIMZ). See "Invalid Operation Exception" on page 1-40 for information about this register.
12	Floating-Point Invalid Operation Exception (Invalid Compare) (VXVC). See "Invalid Operation Exception" on page 1-40 for information about this register.
13	Floating-Point Fraction Rounded (FR). The last floating-point instruction that rounded the intermediate result incremented the fraction.
14	Floating-Point Fraction Inexact (FI). The last floating-point instruction that rounded the intermediate result produced an inexact fraction or a disabled exponent overflow.

Bit	Description
15–19	Floating-Point Result Flags (FPRF).

Bit	Description
15	Floating-Point Result Class Descriptor (C)
16–19	Floating-Point Condition Code (FPCC).

Bit	Description
16	Floating-point less than or negative (FL or <)
17	Floating-point greater than or positive (FG or >)
18	Floating-point equal or zero (FE or equals)
19	Floating-point unordered or NaN (FU). Floating-point compare instructions always set one of the FPCC bits to 1 and the other three FPCC bits to 0. Other instructions can set the FPCC bits with the C bit to encode these 5 bits to indicate the class of the stored result. See the following table for the floating-point result flags. Notice that in this case the three high-order bits of the FPCC retain their relational significance indicating that the value is less than, greater than, or equal to zero.

Floating-Point Result Flags	
Result Flags C < > = ?	Result Value Class
10001	– Quiet NaN
01001	– Infinity
01000	– Normalized number
11000	– Denormalized number
10010	– Zero
00010	+ Zero
10111	+ Denormalized number
00100	+ Normalized number
00101	+ Infinity

Bit	Description
20–21	Reserved.
22	Floating-Point Invalid Square Root Exception. See "Invalid Operation Exception" on page 1-40 for information about this register.
23	Floating-Point Invalid Integer Convert Exception. See "Invalid Operation Exception" on page 1-40 for information about this register.
24	Floating-Point Invalid Operation Exception Enable (VE). See "Invalid Operation Exception" on page 1-40 for information about this register.
25	Floating-Point Overflow Exception Enable (OE). See "Overflow Exception" on page 1-42 for information about this register.
26	Floating-Point Underflow Exception Enable (UE). See "Underflow Exception" on page 1-44 for information about this register.
27	Floating-Point Zero Divide Exception Enable (ZE). See "Zero Divide Exception" on page 1-41 for information about this register.
28	Floating-Point Inexact Exception Enable (XE). See "Inexact Exception" on page 1-44 for information about this register.
29	Reserved.
30–31	Floating-Point Rounding Control (RN). See "Rounding" on page 1-37 for information about this register.

Setting	Description
00	Round to Nearest
01	Round toward Zero
10	Round toward +Infinity
11	Round toward –Infinity.

Note: Every exception bit in the FPSCR is sticky (bits 0 to 12) except the Floating-Point Enabled Exception Summary and Floating-Point Invalid Operation Exception Summary bits. That is, once set they remain set until one of the following instructions possibly changes them: mtfsf, mtfsfi, mtfsb0, and mcrfs.

Floating-Point Data Representation

This section describes how data is represented in the Floating-Point Processor.

Data Format

This architecture defines the representation of a floating-point value in two different binary fixed-length formats. The format can be a one-word format for a single-precision floating-point value or a two-word format for a double-precision floating-point value. The single format (see Figure 8) can be used for data in memory. The double format (see Figure 9) can be used for data in memory and for data in Floating-Point registers. The length of the exponent and the fraction fields differ between these two formats.

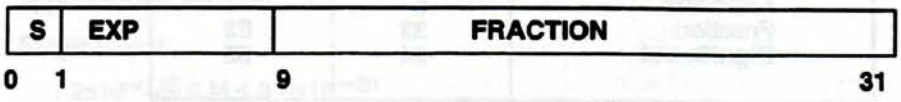


Figure 8. Floating-Point Single Format

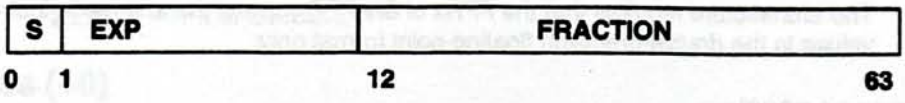


Figure 9. Floating-Point Double Format

Values in floating-point format are composed of the following fields:

Field	Description
S	Sign bit
EXP	Exponent + Bias
FRACTION	Fraction.

Bit 0 is the Sign bit. The xMSB bit is the most significant bit of the EXP field, the xLSB bit is the least significant bit of the EXP field. The fMSB bit is the most significant bit of the FRACTION field. The fLSB bit is the least significant bit of the FRACTION field.

Representation of numerical values in the floating-point formats consist of a Sign bit S, a biased exponent EXP, and the fraction portion FRACTION, of the significand. The significand consists of a Leading Implied bit concatenated on the right with the FRACTION field. This Leading Implied bit is a 1 for normalized numbers and a 0 for denormalized numbers and is located in the Unit bit position (the first bit to the left of the binary point).

Values represented within the two floating point formats can be specified by the parameters listed in Figure 10.

Parameter	Format	
	Single	Double
Exponent bias	+127	+1023
Maximum exponent	+127	+1023
Minimum exponent	−126	−1022
Widths (bits)		
Format	32	64
Sign	1	1
Exponent	8	11
Fraction	23	52
Significand	24	53

Figure 10. IEEE Floating-Point Fields

The architecture requires that the FPRs of the FPP support the arithmetic instructions on values in the double-precision floating-point format only.

Value Representation

This architecture defines numerical and nonnumerical values representable within each of the two supported formats. The numerical values are approximations to the real numbers and include the normalized numbers, denormalized numbers, and zero values. The nonnumerical values representable are the infinities and the NaN values. The infinities are adjoined to the real numbers but are not numbers themselves, and the standard rules of arithmetic do not hold when they appear in an operation. They are related to the real numbers by order alone. Restricted operations among numbers and infinities can be defined. Figure 11 shows the relative location on the real number line for each of the defined entities.

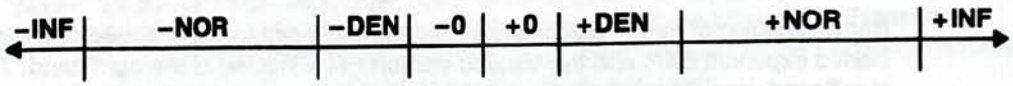


Figure 11. Approximation to Real Numbers

The NaN values are not related to the numbers or infinities by order or value, but are encodings used to convey diagnostic information such as the representation of uninitialized variables.

The following sections describe the different floating-point values defined in the architecture.

Binary Floating-Point Numbers

Machine-representable values are used as approximations to real numbers. Three categories of numbers are supported: normalized numbers, denormalized numbers, and zero values.

Normalized Numbers (+NOR)

The following are values that have a biased exponent value in the range:

- 1 to 254 in single format
- 1 to 2046 in double format.

They are values in which the implied Unit bit is 1. Normalized numbers are interpreted as follows:

NOR equals $(-1)^{**s} \times 2^{**E} \times (1.\text{fraction})$

where *s* is the sign, *E* is the unbiased exponent, and *1.fraction* is the significand that is composed of a leading Unit bit (implied bit) and a fraction part.

The ranges covered by the magnitude (M) of a normalized floating-point number are approximately equal to:

Single format:

$1.2 \times 10^{**-38} \leq M \leq 3.4 \times 10^{**38}$

Double format:

$2.2 \times 10^{**-308} \leq M \leq 1.8 \times 10^{**308}$

Zero Values (+0)

Zero values are values that have a biased exponent value of 0 and a fraction value of 0. Zeros can have a positive or negative sign.

Denormalized Numbers (+DEN)

Denormalized numbers are values that have a biased exponent value of 0 and a nonzero fraction value. They are nonzero numbers smaller in magnitude than the representable normalized numbers. They are values in which the implied Unit bit is 0. Denormalized numbers are interpreted as follows:

DEN equals $(-1)^{**s} \times 2^{**Emin} \times (0.\text{fraction})$

where *Emin* is the minimum representable exponent value (–126 for single precision, –1022 for double precision).

Infinities (+INF)

Infinities are values that have the maximum biased exponent value of:

- 255 in the single format
- 2047 in the double format.

The fraction value of an infinity is zero. They are used to approximate values greater in magnitude than the maximum normalized value.

Infinity arithmetic is defined as the limiting case of real arithmetic, with restricted operations defined among numbers and infinities. Infinities and the real numbers can be related by ordering in the affine sense:

–INF < every finite number < + INF

Arithmetic on infinities is exact and usually does not signal an exception. Exceptions occur because of invalid operations. See "Invalid Operation Exception" on page 1-40 for information.

Not a Numbers (NaNs)

NaN values are values that have the maximum biased exponent value and a nonzero fraction value. The Sign bit is ignored (NaN values are neither positive nor negative). If the high-order bit of the fraction field is 1, it is defined as a *quiet* NaN (QNaN); otherwise, it is defined as a *signaling* NaN. Quiet NaNs are used to represent the result of certain invalid operations. When the Invalid Operation Exception is disabled, FPSCR(VE) equals 0. Examples include undefined arithmetic operations on infinities or NaNs. NaNs used in this manner can convey diagnostic information to help identify results from these invalid operations. Signaling NaNs are used to signal exceptions when they appear as arithmetic operands, while quiet NaNs propagate through most operations without signaling exceptions regardless of the condition of the operation. Specific encoding can thus be preserved through a number of arithmetic operations for its intended use as diagnostic information. When a QNaN is the result of an operation because one of the operands is a NaN or because a QNaN was generated due to a disabled Invalid Operation Exception, then the following rule is applied to determine the NaN with the High-Order Fraction bit set to 1 that is to be stored as the result.

```
If (FRA) is a NaN
    Then (FRT) ← (FRA)
    Else if (FRB) is a NaN
        Then (FRT) ← (FRB)
        Else if (FRC) is a NaN
            Then (FRT) ← (FRC)
            Else if generated QNaN
                Then (FRT) ← generated QNaN
```

If the operand specified by the FRA is a NaN, that NaN is stored as the result. If the operand specified by the FRB is a NaN (if the instruction specifies an FRB operand), that NaN is stored as the result. If the operand specified by the FRC is a NaN (if the instruction specifies an FRC operand), that NaN is stored as the result. If a QNaN was generated due to a disabled Invalid Operation Exception, that QNaN is stored as the result. If a QNaN is to be generated as a result, the QNaN generated has a Sign bit of 0, an exponent field of all ones and a High-Order Fraction bit of 1 with all other fraction bits 0. Any instruction that generates a QNaN as the result of a disabled Invalid Operation generates this QNaN.

Normalization and Denormalization

When an arithmetic operation produces an intermediate result, consisting of a Sign bit, an exponent, and a nonzero significand with a 0 leading bit, it is not a normalized number and must be normalized before it is stored.

To normalize a number, the significand is shifted left while the exponent is decremented by one for each bit shifted, until the leading significand bit becomes 1. The Guard bit and the Round bit (See "Execution Model for IEEE Operations" on page 1-45) participate in the shift with zeros shifted into the Round bit. The exponent is regarded as if its range were unlimited. If the resulting exponent value is less than the minimum value that can be represented in the format specified for the result, the intermediate result is said to be *Tiny*. The stored result is determined by the rules described in "Underflow Exception" on page 1-44. The sign of the number does not change.

When an arithmetic operation produces a nonzero intermediate result with an exponent value less than the minimum value that can be represented in the format specified for the result, the stored result is determined by the rules described in "Underflow Exception" on page 1-44. This process may require denormalization.

To denormalize a number, the significand is shifted right while the exponent is incremented by one for each bit shifted until the exponent is equal to the format minimum value. If any significant bits are lost in this shifting process then *Loss of Accuracy* has occurred and *Underflow Exception* is signaled. See "Underflow Exception" on page 1-44 for more information. The sign of the number does not change.

When denormalized numbers are operands of multiply and divide operations they are prenormalized internally before the operations are performed.

Precision

All arithmetic operations are performed in floating-point double-precision. Floating-point single-precision is obtained with the implementation of four forms of instructions:

1. Load Floating-Point Single

   This form of instruction accesses a single-precision operand in memory, converts it to double-precision operand, and loads it into an FPR. No exceptions are detected on the load operation.

2. Arithmetic operation performed in double precision

3. Round to Floating-Point Single

   This form of instruction rounds a double-precision operand to single-precision, checks the exponent for single-precision range, handles any exceptions according to respective enable bits, and stores that operand into an FPR as a double-precision operand.

4. Store Floating-Point Single

   This form of instruction converts a double-precision operand to single-precision and stores that operand into memory. If the operand requires denormalization in order to fit in single-precision, it is denormalized prior to storing it. No exceptions are detected on the store operation. (Assumes step 3 has been executed.)

Rounding

All arithmetic instructions defined by this architecture produce an intermediate result that can be regarded as being infinitely precise. This result must then be written with a precision of finite length into an FPR. After normalization or denormalization, if the infinitely precise intermediate result is not representable, it must be rounded.

Four modes of rounding are provided that are user-selectable through the Floating-Point Rounding Control field in the FPSCR. These are encoded as follows:

RN	Rounding Mode
00	Round to Nearest
01	Round towards Zero
10	Round towards +Infinity
11	Round towards –Infinity.

Let *Z* be the infinitely precise intermediate arithmetic result or the operand of a convert operation. If *Z* can be represented exactly in the target format, rounding in all modes is equivalent to truncation of *Z*. If *Z* cannot be represented exactly in the target format, let *Z1* and *Z2* be the next largest and next smallest numbers representable in the target format that

bound *Z*, then *Z1* or *Z2* can be used to approximate the result in the target format. Figure 12 shows the relation of *Z*, *Z1*, and *Z2*.

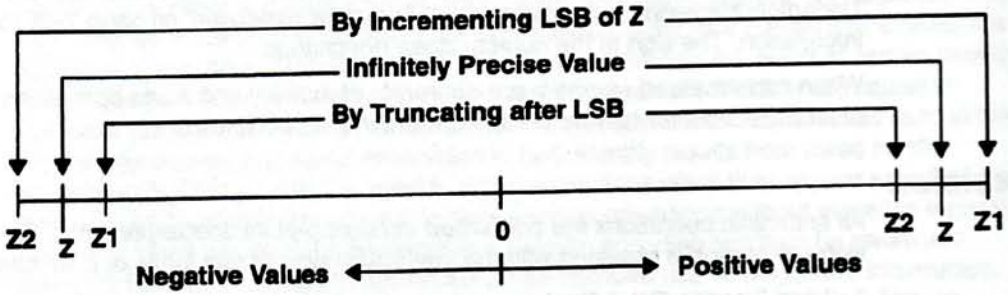


Figure 12. Selection of Z1 and Z2

The following rules specify the rounding in the four modes:

Round To Nearest	Choose the best approximation of Z1 or Z2. In case of a tie, choose the one that is even (least significant bit 0).
Round Toward Zero	Choose the smaller in magnitude (Z1 or Z2).
Round Toward +Infinity	Choose Z1.
Round Toward –Infinity	Choose Z2.

The arithmetic instructions are defined for operations on values that are in the double format.

See "Execution Model for IEEE Operations" on page 1-45 for a detailed explanation of rounding.

Data Handling

Instructions are defined to move floating-point data between the FPRs and memory. For double format the data is not altered during the move. For single-format data, a format conversion from single to double is performed when loading from memory into an FPR and a format conversion from double to single is performed when storing from an FPR to memory. No floating-point exceptions are raised during these operations.

The arithmetic instructions interpret the operand data and produce result data only in the double format.

Note: The Round Floating-Point Double to Single instruction is provided to allow value conversion from double to single precision with appropriate exception checking and rounding. This instruction should be used after every arithmetic operation for obtaining conforming IEEE single-precision results.

Floating-Point Exceptions

This architecture defines the following floating-point exceptions:

- Invalid Operation Exception
  - SNaN
  - Infinity – Infinity
  - Infinity x Zero
  - Infinity ÷ Infinity
  - Zero ÷ Zero
  - Ordered Compare with a NaN
- Zero Divide Exception
- Overflow Exception
- Underflow Exception
- Inexact Exception.

These exceptions can occur during the floating-point arithmetic and conversion operations. For each exception, there is one FPSCR bit to indicate occurrence of the exception and another FPSCR bit to indicate whether the exception is enabled or disabled. If any of these exceptions are recognized during the execution of a floating-point instruction, the exception condition is signalled by setting the corresponding exception bit for the condition in the FPSCR. A Floating-Point Exception Summary bit in the FPSCR is set when any of the exception bits changes from 0 to 1, or when explicitly set by software. A Floating-Point Enabled Exception Summary bit in the FPSCR is set when any of the exceptions are set and the exception is enabled (enable bit is 1).

Multiple exceptions can be set in four cases:

- Inexact Exception can be set with Overflow Exception.
- Inexact Exception can be set with Underflow Exception.
- Invalid Operation Exception (SNaN) can be set with Invalid Operation Exception (Inf x 0) for multiply-add type instructions.
- Invalid Operation Exception (SNaN) can be set with Invalid Operation Exception (NaN Compare) for compare instructions.

When an exception occurs, a result can be delivered or the instruction execution can be suppressed depending on the exception. When a result is to be delivered, it can be a different value for the enabled and disabled conditions for some of the exceptions.

The IEEE standard specifies the handling of the exceptional conditions in terms of traps and trap handlers. In this architecture, an Exception Enable bit of 1 causes the generation of result values as specified in the IEEE standard for the *trap enabled* case. An Exception Enable bit of 0 causes the generation of *default result* values as specified for the trap disabled (or *no trap occurs* or *trap is not implemented*) case. The result to be delivered in each case for each exception is described in the following sections.

In this architecture the detection of the floating-point exception conditions requires either a programmed test or enabling of program interrupts to be generated on enabled floating-point exceptions. For the programmed test to uniquely and precisely detect all exceptions that occur, each instruction that can cause a floating-point exception should be followed by a software branch to a handling routine. For program Interrupt detection, MSR(FE) or MSR(IE) must be set to one and the desired floating-point exception enable bits must also be set to ones.

If MSR(FE) is a one and a floating-point operation causes an enabled exception, a precise PRogram Interrupt is generated. For a precise interrupt, the address saved in SRR0 is the address of the instruction that caused the interrupt, all instructions prior to the instruction casing the exception have completed, and no instruction subsequent to the instruction causing their exception has been executed. A Floating-Point Imprecise Interrupt is generated when MSR(FE) is a zero, MSR(IE) is a one, and a floating-point operation causes an enabled exception. For an imprecise interrupt, some number of instructions beyond the instruction causing the exception may have been executed and the address saved in SRR0 points to an instruction that has not been executed.

Note: This program interrupt is generated every cycle that FPSCR(FEX) equals 1 and MSR(FE) equals 1. It is the responsibility of the exception handler to clear the exception bit that caused the interrupt. Also, the address of the instruction that causes the interrupt is the address that is saved in the SRR 0 register, and, if the SRR 0 register is unaltered, that instruction is the instruction returned to and re-executed. For certain types of floating-point exceptions, returning to the instruction following the instruction that caused the interrupt may be required; therefore, the exception handler is required to increment the address in the SRR 0 register by 4.

System performance with the MSR(FE) bit set to 1 can be significantly degraded.

Floating-Point Exception bits in the FPSCR are sticky. That is, once set, they remain set until software resets them with either a mtfsf, mtfsfi, mtfsb1, mtfsb0, or mtcrfs instruction.

Instruction execution is suppressed in some cases when an exception occurs, so there is no possibility that one of the operands would be lost. These cases are:

- Enabled Invalid Operation
- Enabled Zero Divide.

In all other cases, a specified result is generated and written to the destination specified for the instruction causing the exception. These cases are:

- Disabled Invalid Operation
- Disabled Zero Divide
- Disabled Overflow
- Disabled Underflow
- Disabled Inexact
- Enabled Overflow
- Enabled Underflow
- Enabled Inexact.

The following sections define each of the floating-point exceptions and specify the action to be taken when they are detected. For single-precision applications, the exception detection and handling can be slightly different. See the Floating Round to Single Precision instruction in the *Assembler Language Reference* for exceptions and handling of exceptions for single-precision floating-point arithmetic.

Invalid Operation Exception

Definition

An *Invalid Operation Exception* occurs when an operand is invalid for the specified operation. The invalid operations follow:

- Any operation on a signaling NaN (SNaN)
- For add or subtract operations, magnitude subtraction of infinities (INF – INF)
- Multiplication of zero by infinity (INF x 0)
- Division of zero by zero (0 + 0)
- Division of infinity by infinity (INF + INF)

- Ordered comparison involving a NaN (NaN Compare)
- Square Root of a number that is both negative and nonzero (Invalid Square Root)
- Integer conversion of a NaN or a number that is too large (Invalid Integer Conversion).

Action

The action to be taken depends on the setting of the Invalid Operation Exception Enable bit of the FPSCR.

When the Invalid Operation Exception Enable bit is enabled, FPSCR(VE) equals 1, and invalid operation occurs, the following actions are taken:

1. Instruction execution is suppressed; operands are unmodified.
2. One of the following invalid operation exceptions is set:

FPSCR(VXSNAN)	(if SNaN)
FPSCR(VXISI)	(if INF – INF)
FPSCR(VXIDI)	(if INF ÷ INF)
FPSCR(VXZDZ)	(if 0 ÷ 0)
FPSCR(VXIMZ)	(if INF x 0)
FPSCR(VXVC)	(if NaN Compare)
FPSCR(VXSQRT)	(if Invalid Square Root)
FPSCR(VXCVI)	(if Invalid Integer Convert).

3. If the operation is a compare operation, the FPCC field is set to reflect the Floating-Point Unordered bit. Refer to the "Floating-Point Status and Control Register" on page 1-29 for more details.

When the Invalid Operation Exception Enable bit is disabled, FPSCR(VE) equals 0, and invalid operation occurs, the following actions are taken:

1. One of the invalid operation exceptions is set:

FPSCR(VXSNAN)	(if SNaN)
FPSCR(VXISI)	(if INF – INF)
FPSCR(VXIDI)	(if INF ÷ INF)
FPSCR(VXZDZ)	(if 0 ÷ 0)
FPSCR(VXIMZ)	(if INF x 0)
FPSCR(VXCVI)	(if Invalid Integer Convert).

2. If the operation destination is an FPR, the result is a QNaN.
3. If a result is generated, the FPRF field in the FPSCR is set to reflect the quiet NaN result. If the operation is a compare operation, the FPCC field is set to reflect the Floating-Point Unordered bit. Refer to the "Floating-Point Status and Control Register" on page 1-29 for more details.

Zero Divide Exception

Definition

A *Zero Divide Exception* occurs when a divide instruction is executed with a zero divisor value and a finite nonzero dividend value.

Action

The action taken depends on the setting of the Zero Divide Exception Enable bit of the FPSCR.

When the Zero Divide Exception Enable bit is enabled, FPSCR(ZE) equals 1, and a zero divide exception occurs, the following actions are taken: FPSCR(ZX) ← 1. (A value of 1 is stored in the 2x bit of the FPSCR.

1. Instruction execution is suppressed; operands are unmodified.
2. The Zero Divide Exception bit is set: FPSCR(ZX) ←1.

When the Zero Divide Exception Enable bit is disabled, FPSCR(ZE) equals 0, and a zero divide exception occurs, the following actions are taken:

1. The Zero Divide Exception bit is set: FPSCR(ZX) ← 1.
2. The result is set to ± infinity, where the sign is determined by the exclusive 'OR' of the sign of the operands.
3. The FPRF field in the FPSCR is set to indicate an infinity with the proper sign.
4. The result is placed into the target FPR.

Overflow Exception

Definition

An *overflow* occurs when the magnitude of what would have been the rounded result, if the exponent range were unbounded, exceeds the magnitude of the largest finite number of the specified result precision.

Action

The action to be taken depends on the setting of the Overflow Exception Enable bit of the FPSCR.

When the Overflow Exception Enable bit is enabled, FPSCR(OE) equals 1, and exponent overflow occurs, the following actions are taken:

1. The Overflow Exception is set: FPSCR(OX) ← 1.
2. The exponent of the normalized intermediate result is adjusted by subtracting 1536.
3. The FPRF field in the FPSCR is set to indicate a normalized number with the proper sign.
4. The rounded result is placed into the specified FPR.

When the Overflow Exception Enable bit is disabled, FPSCR(OE) equals 0, and overflow occurs, the following actions are taken:

1. The Overflow Exception bit is set: FPSCR(OX) ← 1.
2. The Inexact Exception bit is set: FPSCR(XX) ← 1.
3. The result is determined by the rounding mode, FPSCR(RN), and the sign of the intermediate result as follows: for negative overflows, store –Infinity; and, for positive overflows, store the format's largest finite number.
   a. Round To Nearest: Store ± Infinity, where the sign is the sign of the intermediate result.
   b. Round To Zero: Store the format's largest finite number with the sign of the intermediate result.

c. Round To + Infinity: For negative overflows, store the format's most negative finite number, and, for positive overflows, store + infinity.

d. Round To – Infinity: For negative overflows, store – infinity and, for positive overflows, store the format's largest finite number.

4. The FPRF field in the FPSCR is set to indicate the class and sign of the result.

5. The result is placed into the specified FPR.

FPA 2.4 Implementation Note

An Overflow Exception applies to machines with an FPA level of 2.4. To identify the level of the FPA, run the **lscfg** command with a **–v** flag. This produces a list of the vital product data. Under the processor component near the beginning of the list, there is a line similar to the following:

```
Device Specific. (Z0).....01XXyy
```

This exception applies only if the XX equals a value of 24.

Overflow occurs when the magnitude of the rounded intermediate result exceeds that of the largest finite number of the specified result precision.

The Floating Round to Single Precision instruction may produce incorrect results when all the following conditions are met:

- The Floating Round to Single Precision instruction is dependent on a previous floating-point arithmetic operation. Dependent means that it uses the target register of the arithmetic operation as the source register.
- Less than two nondependent floating-point arithmetic operations occur between the Floating Round to Single Precision instruction and the operation on which it is dependent.
- The magnitude of the double-precision result of the arithmetic operation is less than 2**128 before rounding.
- The magnitude of the double-precision result after rounding is exactly 2**128.

Resultant Value

If the error occurs, the magnitude of the result placed in the target register is 2**128:

X'47F0000000000000' or X'C7F0000000000000'

This is not a valid single precision value. The setting of the FPSCR and Condition register (CR) will be the same as if the result did not overflow.

Insuring Correct Results

If after considering the results described previously, the programmer decides that the error will cause significant problems for the application, either of the following methods may be used to avoid the error:

- Ensure that two nondependent floating-point operations are placed between a floating-point arithmetic operation and the dependent round to single. The target register for these operations should not be the same register that the Floating Round to Single Precision instruction uses as a source register.
- Insert two floating-round-to-single-precision operations when the floating-round-to-single-precision operation may be dependent on an arithmetic operation that precedes it by less than three floating-point instructions.

Either solution degrades performance by an amount dependent on the particular application.

Underflow Exception

Definition

Underflow Exception is defined separately for the *enabled* and *disabled states*:

Enabled: Underflow occurs when the intermediate result is Tiny.

Disabled: Underflow occurs when the intermediate result is Tiny and there is Loss of Accuracy.

A *Tiny* result is detected before rounding when a nonzero result value, computed as though the exponent range were unbounded, would be less in magnitude than the smallest normalized number.

If the intermediate result is Tiny and the Underflow Exception Enable bit is off, FPSCR(UE) equals 0, the intermediate result is to be denormalized and rounded. See "Normalization and Denormalization" on page 1-36 and "Rounding" on page 1-37 for information about denormalizing and rounding results.

Loss of Accuracy is detected as an inexact result when the delivered result value differs from what would have been computed were both the exponent range and precision unbounded.

Action

The action to be taken depends on the setting of the Underflow Exception Enable bit of the FPSCR.

When the Underflow Exception Enable bit is enabled, FPSCR(UE) equals 1, and exponent underflow occurs, the following actions are taken:

1. The Underflow Exception bit is set: FPSCR(UX) ← 1.
2. The exponent of the normalized intermediate result is adjusted by adding 1536.
3. The FPRF field in the FPSCR is set to indicate a normalized number with the proper sign.
4. The rounded result is placed into the specified FPR.

Note: The FR and FI bits in the FPSCR allow the trap handler to simulate a trap disabled environment. The bits provide enough information to unround the result prior to denormalization.

When the Underflow Exception Enable bit is disabled, FPSCR(UE) equals 0, and underflow occurs, the following actions are taken:

1. The Underflow Exception bit is set: FPSCR(UX) ← 1.
2. The FPRF field in the FPSCR is set to indicate the class and sign of the result (±Denormalized Number or ±zero).
3. The rounded result is placed into the specified FPR.

Inexact Exception

Definition

The *Inexact Exception* occurs when one of two conditions occurs during rounding:

- The rounded result differs from the intermediate result assuming the intermediate result exponent range and precision to be unbounded.
- The rounded result overflows and the Overflow Exception is disabled.

Action

When the Inexact Exception occurs, the following actions are taken:

1. The Inexact Exception bit is set: FPSCR(XX) ← 1.
2. The FPRF field in the FPSCR is set to indicate the class and sign of the result.
3. The rounded or overflowed result is placed into the destination FPR.

Floating-Point Resource Management

Facilities are defined to allow control of the use of the Floating-Point Processor. MSR(FP) is the Floating-Point Available bit. It controls the execution of floating-point instructions. When the FPP is available, MSR(FP) equals 1 and the floating-point instructions can be executed. Otherwise the FPP is unavailable, and MSR(FP) equals 0. An attempt to execute a floating-point instruction in this state causes a Floating-Point Unavailable Interrupt and the instruction execution is suppressed.

The test for an invalid processor op code is made before the MSR(FP) bit is inspected.

Floating-Point Execution Models

All implementations of this architecture must provide the equivalent of the following execution models to ensure that identical results are obtained.

Special rules are provided in the definition of the arithmetic instructions for the infinities, denormalized numbers, and NaNs.

Although the double-precision format specifies an 11-bit exponent, exponent arithmetic makes use of two additional bit positions to avoid potential transient overflow conditions. One extra bit is required when denormalized double-precision numbers are prenormalized. The second bit is required to permit the computation of the adjusted exponent value in each of the following cases when the corresponding Exception Enable bit is 1:

- Underflow during multiplication using a denormalized factor.
- Overflow during division using a denormalized divisor.

Execution Model for IEEE Operations

IEEE conforming significand arithmetic is considered to be performed with a floating-point accumulator. Figure 13 shows the format of the accumulator.

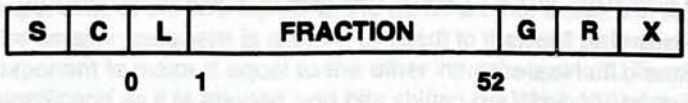


Figure 13. IEEE Execution Model

Field	Description
S	Sign bit
C	Carry bit that captures the carry out of the significand
L	Leading Unit bit of the significand that receives the Implicit bit from the operands
FRACTION	Fraction, a 52-bit field that accepts the fraction of the operands.

The Guard (G), Round (R), and Sticky (X) bits are extensions to the low-order bits of the accumulator. The G and R bits are required for post normalization of the result. The G, R, and X bits are required during rounding to determine if the intermediate result is equally near the two nearest representable values. The X bit serves as an extension to the G and R bits by representing the logical 'OR' of all bits that can appear to the low-order side of the R bit, either due to shifting the accumulator right or other generation of low-order result bits. The G and R bits participate in the left shifts with zeros being shifted into the R bit. Figure 14 shows the significance of the G, R, and X bits with respect to the intermediate result (IR), the next lower in magnitude representable number (NL), and the next higher in magnitude representable number (NH).

G R X	Interpretation
0 0 0	IR is exact
0 0 1 0 1 0 0 1 1	IR closer to NL
1 0 0	IR midway between NL and NH
1 0 1 1 1 0 1 1 1	IR closer to NH

Figure 14. Interpretation of G, R, and X Bits

The significand of the intermediate result is made up of the L bit, the FRACTION field, and the G, R, and X bits.

The infinitely precise intermediate result of an operation is the result normalized in the L, FRACTION, G, R, and X bits of the floating-point accumulator.

Before the results are stored into an FPR, the significand is rounded using the rounding mode specified by the Floating-Point Rounding Control field (RM) of the FPSCR. If rounding results in a carry into the C bit, the significand is shifted right one position and the exponent is incremented by one. This, in turn, can result in an exponent overflow. Fraction bits to the left of the bit position used for rounding are stored in the FPR, and low-order bit positions, if any, are set to 0.

Four modes of rounding are provided that are user-selectable through the Floating-Point Rounding Control field (RM) of the FPSCR. This field is encoded as follows:

RN	Rounding Mode
00	Round To Nearest
01	Round Toward Zero
10	Round Toward + Infinity
11	Round Toward – Infinity

For rounding, the conceptual Guard, Round, and Sticky bits are defined in terms of accumulator bits. The following table refers to the bit positions of Guard, Round, and Sticky for double and single-precision FP numbers.

Location of the Guard, Round, and Sticky Bits			
Format	**Guard**	**Round**	**Sticky**
Double	G bit	R bit	X bit
Single	24	25	26–52 G, R, X

Rounding can be treated as though the significand were shifted right, if required, until the least significant bit to be retained is in the low-order bit position of the FRACTION field. If any of the Guard, Round, or Sticky bits are nonzero, the result is inexact.

Z1 and Z2, as defined in "Rounding," on page 1-37 can be used to approximate the result in the target format when one of the following rules is used.

If rounding results in a carry into the C bit, the significand must be shifted right one position and the exponent is increased by one. This can result in signaling an inexact result if the low order bit of the fraction had been a 1.

Where the result is to have fewer than 53 bits of precision because the instruction is a round to single-precision, the intermediate result is either normalized, or is placed in correct denormalized form before the result is rounded.

Execution Model for Multiply-Add Type Instructions

The architecture makes use of a special form of instruction that performs up to three operations in one instruction (a multiply, an add, and a negate operation). With this added capability is the special feature of being able to produce a more exact intermediate result as an input to the rounder. Figure15 shows the intermediate results produced by the multiply-add operations.

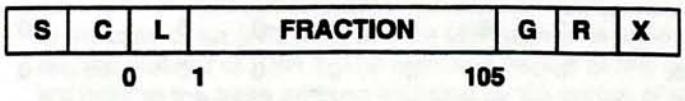


Figure 15. Multiply Add Execution Model

The first part of the operation is a multiply operation. The multiply operation has two 53-bit significands as inputs, which are assumed to be prenormalized, and produces a result conforming to the preceding model. The sign produced by the multiply operation portion is defined to be the XOR of the signs of the two multiply input operands. If there is a carry out of the significand (C), the significand is shifted to the right by one bit, shifting the L bit (Leading Unit bit) into the most significant bit of the fraction, shifting the C bit (carry out) into the L bit. All 106 bits (L bit, the fraction) of the product take part in the add operation. If the exponents of the two inputs to the adder are not equal, the significand of the operand with the smaller exponent is aligned (shifted) to the right by an amount that is added to that exponent to make it equal to the other inputs' exponent. Zeros are shifted into the left of the significand as it is aligned and bits shifted out of bit 105 of the significand are ORed into the X bit. The add operation also produces a result conforming to the preceding model with the X bit taking part in the add operation. The sign produced by the add portion is defined to be the sign of the largest of the two add input operands. When the sum of two operands with opposite signs is exactly zero, the sign of that sum is positive in all rounding modes except Round Toward – Infinity, in which mode that sign is negative. The sum of operands with the same sign retains the sign of the operands, even if the operands are zeros.

The result of the add is then normalized, with all bits of the add result, except the X bit, participating in the shift. The normalized result provides an intermediate result as input to

the rounder that conforms to the model described in "Execution Model for IEEE Operations." on page 1-45 The intermediate result has the following characteristics:

- The Guard bit is bit 53 of the intermediate result.
- The Round bit is bit 54 of the intermediate result.
- The Sticky bit is the OR of all remaining bits to the right of bit 55, inclusive.

The rules of rounding the intermediate result are the same as the described in "Execution Model for IEEE Operations" on page 1-45.

If the instruction is Floating Negative Multiply Add or Floating Negative Multiply Subtract, the negation occurs after rounding.

Interrupts

This section describes the function of and control over the System Interrupt mechanism. Except for the Supervisor Call Interrupt, an interrupt is composed of the following actions:

1. Loading SRR 0 with the address of the current or the next instruction (bits 30 and 31 are 0).
2. Loading bits 0 to 15 of SRR 1 with information specific to each interrupt.
3. Loading bits 16 to 31 of SRR 1 from bits 16 to 31 of the MSR.
4. Setting the MSR according to the following table.

Machine State Register Setting Due to Interrupt									
Interrupt Type	**EE**	**PR**	**FP**	**ME**	**FE**	**AL**	**IP**	**IR**	**DR**
System Reset	0	0	0	N	0	0	N	0	0
Machine Check	0	0	0	0	0	0	N	0	0
Data Storage	0	0	0	N	0	0	N	0	0
Instruction Storage	0	0	0	N	0	0	N	0	0
Alignment	0	0	0	N	0	0	N	0	0
Program	0	0	0	N	0	0	N	0	0
External	0	0	0	N	0	0	N	0	0
FP Unavailable	0	0	0	N	0	0	N	0	0
Trace	0	0	0	N	0	0	N	0	0
FP Imprecise	0	0	0	N	0	0	N	0	0
Supervisor Call	0	0	N	N	0	N	N	N	N

The preceding table uses the following representations:

Setting	Decode
0	Bit is set to 0.
N	Bit is not altered.

5. Beginning the instruction fetch and execution operations using the new MSR value at a location specific to each interrupt type. This location is determined by knowing the base

address, as determined in "Machine State Register" on page 1-22, and by knowing the offset of the interrupt as shown in the following list:

Offset	Interrupt Type
X'00100'	System Reset Interrupt
X'00200'	Machine Check Interrupt
X'00300'	Data Storage Interrupt
X'00400'	Instruction Storage Interrupt
X'00500'	External Interrupt
X'00600'	Alignment Interrupt
X'00700'	Program Interrupt
X'00800'	Floating Unavailable Interrupt
X'00900'	Trace Interrupt (POWER2 only)
X'00A00'	Floating-Point Imprecise Interrupt (POWER2 only)
X'00B00'	Reserved
. .	. . . .
. .	. . . .
X'00F00'	Reserved
X'01000'	Supervisor Call Interrupt
X'01020'	Supervisor Call Interrupt
. .	. . . .
. .	. . . .
X'01FC0'	Supervisor Call Interrupt
X'01FE0'	Supervisor Call Interrupt.

Note: The ranges of memory locations from X'00000B00' to X'00000FFF' and from X'FFF00B00' to X'FFF00FFF' are reserved. Use of these locations risks possible incompatibility with future implementations.

In the case of an SVC Interrupt, the Link register is used instead of SRR 0 and the Count register instead of SRR 1. The execution begins at one of 128 entry points starting at offset X'01000' to the base address indicated by the setting of MSR(IP). In addition, the following bits in the MSR are turned off:

- External Interrupt Enable (EE)
- Problem State (PR)
- FP Exception Interrupt Enable (FE).

The remaining bits are not modified.

Note: Except for the SVC Interrupt, the actions taken at an interrupt include turning off both the instruction and the data translation. Thus, the locations of the first instruction for each of these interrupts are interpreted in a real context. See "Storage Control" on page 1-69 for more information.

All interrupts are precise except the Floating-Point Imprecise, Machine Check and System Reset Interrupts. For Program, Alignment, and Data Storage Interrupts, the address contained in SRR 0 points to the instruction that caused the interrupt. For the Floating-Point Imprecise Interrupt, the address contained in SRR 0 points to an instruction beyond the instruction that caused the interrupt. For External and Instruction Storage Interrupts, the address loaded into SRR 0 points to the instruction that would have executed next. For System Reset and Machine Check Interrupts, the address loaded into SRR 0 points to the instruction currently being executed if that instruction also causes an interrupt; otherwise, it points to the instruction that would have executed next. For SVC Interrupts, the address loaded into the Link register points to the instruction that should be returned to after the SVC Interrupt and does not affect SRR 0.

All instructions prior to the instruction pointed to by SRR 0 (the Link register for SVC) have logically completed at the time of the interrupt and no instruction logically subsequent to it has executed.

In the case of a Data Storage Interrupt or an Alignment Interrupt, neither the RT register in Load instructions nor the RA register in Load/Store with Update instructions are to be altered.

Interrupt Definitions

The following section describes the interrupt definitions for the system processor architecture.

System Reset Interrupt

A System reset begins with a System Reset Interrupt.

The following registers are set as indicated:

SRR 0

Bit	Description
0–31	Set to the address of the instruction currently being executed if that instruction also causes an interrupt; otherwise, set to the address of the instruction that would have executed next.

SRR 1

Bit	Description
0–15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.
28–31	Reserved.

Execution resumes at offset X'00100' from the base address indicated by the setting of MSR(IP).

Machine Check Interrupt

Typical machine failures reported with this interrupt include:

- Instruction Cache Reloads
  - Memory Address Parity Error
  - Uncorrectable ECC Error
  - Address Exception (no extents match)

- Data Cache Reloads
  - Memory Address Parity Error
  - Uncorrectable ECC Error
  - Address Exception (no extents match)
- Data Cache Storebacks
  - Memory Address Parity Error
  - Address Exception (no extents match).

Machine Check Interrupts are enabled when MSR(ME) = 1. If MSR(ME) = 0 and a Machine Check occurs, the Processor Check stops.

The following registers are set as indicated:

SRR 0

Bit	Description
0–31	Set to the address of the instruction currently being executed if the instruction has an exception; otherwise, set to the instruction that would have executed next.

SRR 1

Bit	Description
0–15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Set to 0.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.
28–31	Reserved.

Execution resumes at offset X'00200' from the base address indicated by the setting of MSR(IP).

A Machine Check Interrupt with MSR(ME) = 0 produces a Check Stop condition and the processor halts execution. The machine goes through a self-test and IPL again. See the hardware technical reference manual for your system for more information on the Initial Program Load procedure. Certain hardware failures, such as internal parity errors, hardware hang conditions, and hardware error detection logic can also cause a Check Stop.

Data Storage Interrupt

A Data Storage Interrupt occurs when a data storage access cannot be performed for any of the following reasons:

- I/O Exception, when the execution of a storage access instruction is attempted and the I/O subsystem indicates a failure detected as part of the synchronous execution of that instruction.
- Accessed virtual address cannot be translated.

- Access is a floating load or store to an I/O segment.
- Access violates storage protection.
- Access caused a loop in the hardware translation mechanism.
- Access caused a segment crossing from T = 0 to T = 1.

Such accesses can be generated by load and store type instructions, certain storage controls, and the cache control instructions.

The interrupt cause is defined in a Data Storage Interrupt Status register. These interrupts also use the Data Address register.

The following registers are set as indicated when the interrupt occurs:

SRR 0

Bit	Description
0–31	Set to the address of the failing instruction.

SRR 1

Bit	Description
0–15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.
28–31	Reserved.

DSISR

Bit	Description
0	Set to 1 for an I/O Exception, otherwise to 0.
1	Set to 1 if the end of the selected PTE chain is reached and the translation of an attempted access is not found, otherwise to 0.
2	Set to 1 if a storage access is not permitted by the data-locking mechanism.
3	Set to 1 if a floating load or store instruction references an I/O segment (for example, a segment whose Segment register's T bit equals 1); otherwise, set to 0.

Bit	Description
4	Set to 1 if a storage access is not permitted by the page protection mechanism described in "Page Protection" on page 1-81; otherwise, set to 0.
5	Set to 1 if an access causes a loop in the translation mechanism (for example, the PTE search has gone on for more than 127 attempts)); otherwise, set to 0.
6	Set to 1 for a store operation and to zero for a load operation
7	Set to 1 if a data storage access crosses a segment boundary where the first segment accessed had T = 0 and the segment crossed into has T = 1); otherwise, set to 0.
8–14	Set to 0.
15–31	Undefined.

DAR

Bit	Description
0–31	Set to 1 of the following: • An effective address of a byte in the first word accessed in the page that caused the Data Storage Interrupt, for floating single- and fixed-storage accesses • An effective address of a byte in the first doubleword accessed in the page that caused the Data Storage Interrupt, for floating double accesses • The effective address referenced in the I/O space for I/O exceptions.

Execution resumes at offset X'00300' from the base address indicated by the setting of MSR(IP).

Instruction Storage Interrupt

An Instruction Storage Interrupt occurs when an instruction fetch operation cannot be performed for of any of the following reasons:

- Address cannot be translated.
- Address is in a Special segment.
- Address is in an I/O segment.
- Address is in a protected page.
- Address caused a loop in the hardware translation mechanism.

Such accesses can only be generated by instruction fetch operations. The following registers are set as indicated:

SRR 0

Bit	Description
0–31	Set to the address of the instruction that was being fetched.

SRR 1

Bit	Description
0	Set to 0.
1	Set to 1 if the end of the selected PTE chain is reached and the translation of an attempted access is not found; otherwise, set to 0.
2	Set to 1 if the virtual address used to fetch an instruction is in a Special Segment (SR S bit one); otherwise, set to 0.
3	Set to 1 if the address used to fetch an instruction is in an I/O segment (for example, a segment whose Segment register's T bit equals 1) and Instruction Relocate is on; otherwise, set to 0.
4	Set to 1 if a storage access is not permitted by the page protection mechanism described in "Page Protection" on page 1-81; otherwise, set to 0.
5	Set to 1 if the fetch causes a loop in the translation mechanism (for example, the PTE search has gone on for more than 127 attempts); otherwise, set to 0.
6–15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.
28–31	Reserved.

Execution resumes at offset X'00400' from the base address indicated by the setting of MSR(IP).

Alignment Interrupt

An Alignment Interrupt is raised when MSR(AL) = 1 and one the following conditions is met:

- The effective address generated by a halfword load or store type instruction is not on a halfword storage boundary and the hardware cannot perform the unaligned storage access.
- The effective address generated by a word load or store type instruction is not on a word storage boundary and the hardware cannot perform the unaligned storage access.

- The effective address generated by a doubleword load or store type instruction is not on a doubleword storage boundary and the hardware cannot perform the unaligned storage access.
- The effective address generated by a Load/Store Multiple instruction is not on a word storage boundary.

The following registers are set as indicated. Set the registers to bits 1 to 4 of the instruction if utilizing a D-form instruction.

SRR 0

Bit	Description
0–31	Set to the address of the instruction that caused the interrupt.

SRR 1

Bit	Description
0–15	Set to 0.
16–31	Load from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.
28–31	Reserved.

DSISR

Bit	Description
0–13	Set to 0.
14	Set to value of the T-bit of the Segment register of the storage access that caused the Alignment Interrupt.
15–16	Set to bits 29 to 30 of the instruction if an X-form instruction. Set to bit b'00' if a D-form instruction.
17	Set to bit 25 of the instruction if an X-form instruction. Set to bit 5 of the instruction if a D-form instruction.
18–21	Set to bits 21 to 24 of the instruction if an X-form instruction. Set to bits 1 to 4 of the instruction if a D-form instruction.
22–31	Set to bits 6 to 15 of the instruction.

DAR

Bit	Description
0–31	Set to the effective address that caused the Alignment Interrupt.

Execution resumes at offset X'00600' from the base address indicated by the setting of MSR(IP).

Program Interrupt

A Program Interrupt is generated by any of the following exceptions:

- Floating-Point Enabled Exception

  A Floating-Point Exception Program Interrupt is generated when the MSR(FE) = 1 and the FPSCR(FEX) = 1. FPSCR(FEX) is turned on by the execution of a floating-point instruction that causes an enabled exception or by the execution of a "Move to FPSCR" type instruction which sets both an exception and its corresponding enable.

- Invalid Operation

  An Invalid Operation Program Interrupt is generated when the execution of an instruction is attempted with an undefined opcode or undefined combination of opcode and extended opcode fields.

- Privileged Instruction

  A Privileged Instruction Program Interrupt is generated when the execution of a privileged instruction is attempted and MSR(PR) = 1.

- Trap

  A Trap Program Interrupt is generated when any of the specified set of conditions in a Trap instruction is met.

The following registers are set as indicated:

SRR 0

Bit	Description
0–31	Set to the address of the instruction that caused the Program Interrupt.

SRR 1

Bit	Description
0–10	Set to 0.
11	Set to 1 for a Floating-Point Enabled Exception Program Interrupt; otherwise, to 0.
12	Set to 1 for an Invalid Operation Program Interrupt; otherwise, to 0.
13	Set to 1 for a Privileged Instruction Program Interrupt; otherwise, to 0.
14	Set to 1 for a Trap Program Interrupt; otherwise, to 0.
15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.
28–31	Reserved.

Execution resumes at offset X'00700' from the base address indicated by the setting of MSR(IP).

Note: If FPSCR(FEX) = 1 and MSR(FE) = 0, a Floating-Point Enabled Exception type Program Interrupt can be generated by setting MSR(FE) to one with any instruction that can set the MSR (for example, mtmsr, rfi, and rfsvc). When this occurs, SRR 0 is loaded with the address of the instruction that would have executed next and not to the address of the instruction that modified the MSR causing the interrupt.

External Interrupt

External Interrupts are requested by a signal presented by the External Interrupt mechanism. An External Interrupt occurs when an External Interrupt signal is present and MSR(EE) = 1.

The following registers are set as indicated:

SRR 0

Bit	Description
0–31	Set to the address of the instruction that the processor would have attempted to execute next if no interrupt conditions were present.

SRR 1

Bit	Description
0–15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.

27 DR	Set to 0.
28–31	Reserved.

Execution resumes at offset X'00500' from the base address indicated by the setting of MSR(IP).

Floating-Point Unavailable Interrupt

A Floating-Point Unavailable Interrupt is generated when the execution of any floating-point instruction is attempted and MSR(FP) = 0.

The following registers are set as indicated:

SRR 0

Bit	Description
0–31	Set to the address of the instruction that caused the interrupt.

SRR 1

Bit	Description
0–15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.
28–31	Reserved.

Execution resumes at offset X'00800' from the base address indicated by the setting of MSR(IP).

Trace Interrupt (POWER2 Only)

A Trace Interrupt is generated after every instruction that completes without causing any other interrupt.

The following registers are set as indicated:

SRR 0

Bit	Description
0–31	Set to the address of the instruction to be executed next (next sequential instruction or the instruction that is the target of a taken branch).

SRR 1

Bit	Description
0–15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21 SE	Set to 0.
22 BE	Set to 0.
23 FE	Set to 0.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.
28	Reserved.
29 PM	Set to 0.
30–31	Reserved.

Execution resumes at offset X'00A00' from the base address indicated by the setting of MSR(IP).

Floating-Point Imprecise Interrupt (POWER2 only)

A Floating-Point Imprecise Interrupt is generated when FPSCR(FEX) = 1, MSR(FE) = 0, and MSR(IE) = 1.

The following registers are set as indicated:

SRR 0

Bit	Description
0–31	Set to the address of some instruction beyond the instruction that created the condition where FPSCR(FEX) = 1, MSR(FE) = 0, and MSR(IE) = 1.

SRR 1

Bit	Description
0–15	Set to 0.
16–31	Loaded from bits 16 to 31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Set to 0.
19 ME	Not altered.
20 FE	Set to 0.
21 SE	Set to 0.
22 BE	Set to 0.
23 FE	Set to 0.
24 AL	Set to 0.
25 IP	Not altered.
26 IR	Set to 0.
27 DR	Set to 0.

Bit	Description
28	Reserved.
29 PM	Set to 0.
30–31	Reserved.

Execution resumes at offset X'00A00' from the base address indicated by the setting of MSR(IP).

Supervisor Call Interrupt

An SVC Interrupt occurs when an SVC instruction is executed.

The registers are set as follows:

LR

Bit	Description
0–31	The Link register is set to the address of the instruction following the SVC instruction if LK = 1 in the instruction.

CTR

Bit	Description
0–15	Loaded from bits 16–31 of the SVC instruction.
16–31	Loaded from bits 16–31 of the MSR.

MSR

Bit	Description
0–15	Reserved.
16 EE	Set to 0.
17 PR	Set to 0.
18 FP	Not altered.
19 ME	Not altered.
20 FE	Set to 0.
21–23	Reserved.
24 AL	Not altered.
25 IP	Not altered.
26 IR	Not altered.
27 DR	Not altered.
28–31	Reserved.

If SA = 0, execution resumes at one of 128 entry points starting at offset b'00001' || LEV || b'00000' from the base effective address indicated by the setting of MSR(IP). If SA = 1, execution resumes at offset X'01FE0' from the base effective address indicated by the setting of MSR(IP).

Interrupt Priorities

Interrupts are either unordered or ordered with respect to the Save Restore registers. Machine Check and System Reset Interrupts are unordered. That is, either can occur at any time. When either occurs, the machine immediately changes state according to the rules specified in "System Reset Interrupt" on page 1-50 and "Machine Check Interrupt" on page 1-50. State change is such that any previous interrupt information contained in SRR 0 and SRR 1 is lost. Any other pending interrupt is suppressed. To prevent indefinite looping on System Reset, the interrupt should be viewed as a trigger caused by a system or operator action. The triggering action must be repeated to cause another System Reset. Looping on Machine Check is already prevented by MSR(ME).

The remaining interrupts are ordered, that is, one and only one interrupt can occur at a time. This is due to the serial reusable nature of the SRR 0 and SRR 1 registers. Insuring one and only one interrupt at a time is both a hardware and software responsibility. Hardware must test and present interrupts in the order that follows. Finding an interrupt condition present, the hardware does not continue testing for additional interrupt conditions. Thus, even when there are multiple interruptible conditions present, the hardware does not know about the additional conditions and therefore does not present the associated interrupts. Software, for its part, must save the state of the machine (including SRR 0 and SRR 1) in such a manner that the saving operation does not cause an interrupt.

The Instruction Storage Interrupt is the lowest ordered interrupt. It is generated when the machine is unable to fetch the next instruction.

External Interrupt requires special handling. This is the only maskable ordered interrupt. It is also the lowest ordered interrupt. If the External Interrupt signal is present and is allowed, the hardware cannot present this interrupt until it is determined that no other ordered interrupt condition is present. According to the rules of "Interrupt Definitions," on page 1-50 when any interrupt occurs, External Interrupt is automatically masked. This ensures that an External Interrupt does not immediately follow any other interrupt. After any interrupt, software must not allow an External Interrupt until it has safely saved the state of the machine.

The next higher-ordered interrupt is Instruction Storage Interrupt. This interrupt occurs when the machine is unable to fetch the next instruction. The remaining interrupts are instruction dependent for loads/stores, SVC, Trap, floating, privilege, and undefined instructions. The associated interrupts occur next in the ordering. Each of these types of instructions can only generate one interrupt condition so there is no need for additional ordering in these situations.

For floating load or store instructions, Floating Unavailable Interrupt is ordered higher than Alignment Interrupt, which is ordered higher than Data Storage Interrupt.

For fixed load or store instructions when not accessing an I/O segment, Alignment Interrupt is ordered higher than Data Storage Interrupt.

For fixed load or store instructions when accessing an I/O segment, the only interrupt is an I/O Exception-type Data Storage Interrupt, which can be caused by alignment, privilege, or other I/O conditions. There is no processor architecture definition about the ordering of these conditions.

The following summarizes the interrupts that can be caused due to the direct execution of the listed types of instructions and their relative priority. Not listed are System Reset, Machine Check, Instruction Storage, and External Interrupts.

- For fixed-point loads and stores (T = 0)
  1. Alignment
  2. Data Storage
  3. Trace.
- For fixed-point loads and stores (T = 1)
  1. Data Storage
  2. Trace.
- For floating-point loads and stores (T = 0)
  1. Floating-Point Unavailable
  2. Alignment
  3. Data Storage
  4. Trace.

- For floating-point loads and stores (T = 1)
  1. Floating-Point Unavailable
  2. Data Storage
  3. Trace.

  Note: Some implementations generate an Alignment Interrupt instead of a Data Storage Interrupt in the event the effective address generated is unaligned (not a word or doubleword address). In this case, bit 14 of the DSISR is set to the value of the T bit of the Segment register selected by the effective address.

- For floating-point arithmetic, compare, floating round to single, floating square root, floating covert to integer, and any move to FPSCR instructions
  1. Floating-Point Unavailable
  2. Program (Floating-Point Enabled Exception)
  3. Trace.
- For remaining floating-point instructions
  1. Floating-Point Unavailable
  2. Trace.
- For rfi, rfscv, and mtmsr instructions
  1. Program (Privileged Instruction)
  2. Program (Floating-Point Enabled Exception).
- For trap instructions
  1. Program (Trap).
- Any privileged instructions
  1. Program (Privileged Instruction)
  2. Trace.
- Any undefined instructions
  1. Program (Invalid Operation).

External Interrupt Mechanism for POWER

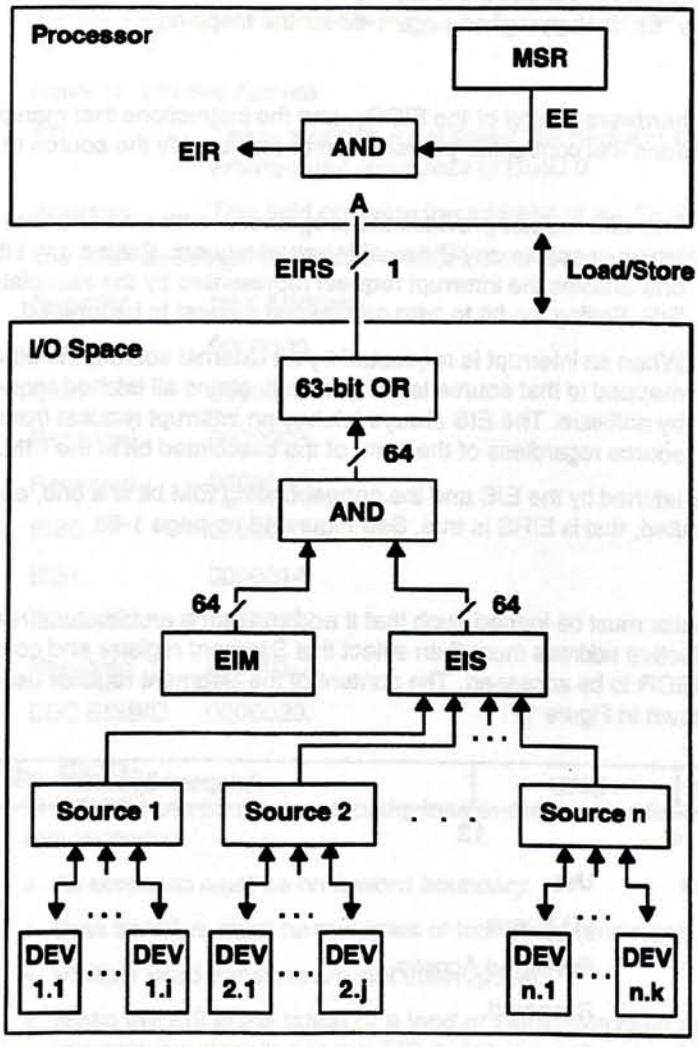
The External Interrupt mechanism provides for the collection and presentation of interrupt requests from external (non-PU) sources. Software and hardware control External Interrupt requests using the following mechanisms:

- External Interrupt Enable MSR(EE).
- External Interrupt Mask (EIM) register (64 bits).
- External Interrupt Summary (EIS) register (64 bits).
- Load and store instructions addressing the EIM, the EIS, and other I/O resources.
- Sources of External Interrupts.

Figure 16 shows the logical structure of the External Interrupt mechanism. This mechanism supports 64 separate External Interrupt sources that are collected into one single External Interrupt Request signal (EIRS). An External Interrupt Request (EIR) is sent to the processor only if the EIRS is present and MSR(EE) = 1.

The EIM register is used as a mask to enable or suppress the requests that have been latched in the EIS register. These registers are referred to as the External Interrupt Control registers (EICR). The EICRs are in the I/O address space in BUID 0. They are manipulated using load register and store register instructions addressing the I/O space.

There is no hardware priority among the bits of the EIS and software can service the requests in any order.



Where:

EIR = External Interrupt Request
EIRS = External Interrupt Request Signal.

Figure 16. External Interrupt Control Mechanism

External Interrupt Enable

MSR(EE) controls the presentation of an external interrupt to the processor. EIR is true only when the following conditions are met:

- MSR(EE) = 1
- One or more enabled External Interrupt requests are pending (the state of EIRS is true).

See "Machine State Register" on page 1-22 for the description of the MSR and MSR(EE).

External Interrupt Control Registers

Both the EIM and the EIS are 64-bit registers in the I/O space. These registers serve different functions but have the same mapping between External Interrupt sources and register bits. See "EICR Mapping" on page 1-66 for the mapping.

Functions

The EICRs, the hardware control of the EICRs, and the instructions that manipulate them provide the functions that control the presentation of and identify the source of External Interrupts.

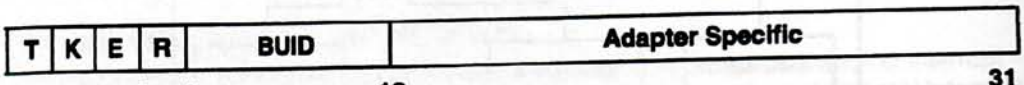
EIM Register The EIM register provides the programmer with a mechanism to selectively inhibit or enable any External Interrupt request. Setting any bit of the EIM to one enables the interrupt request represented by the associated bit in the EIS. Setting the bit to zero causes the request to be ignored.

EIS Register When an interrupt is requested by an external source, the bit of the EIS mapped to that source is set to one. It retains all latched requests until reset by software. The EIS always latches an interrupt request from an external source regardless of the state of the associated bit in the EIM.

If any request is latched by the EIS and the corresponding EIM bit is a one, an External Interrupt is signalled, that is EIRS is true. See Figure 16 on page 1-63.

Addressing the EICRs

A Segment register must be loaded such that it addresses the architectural resources in BUID 0. The effective address must then select this Segment register and contain the address of the EICR to be accessed. The content of the Segment register used to access the EICRs is shown in Figure 17.



Bit	Value	Use
T	b'1'	I/O Space
K	b'0'	Privileged Access
E	b'0'	Reserved
R	b'0'	Reserved
BUID	X'00'	Bus Unit ID 0
AS	X'00000'	Adapter Specific.

Figure 17. Segment Register

Notes:

1. For architectural resources in BUID 0, the adapter specific field contains 0.
2. If the K bit is equal to one and a load or store register instruction is issued addressing the EICRs, an I/O Exception-type Data Storage Interrupt results and no changes are made either to the target or source register.

Figure 18 shows the effective address used to access the EICRs.

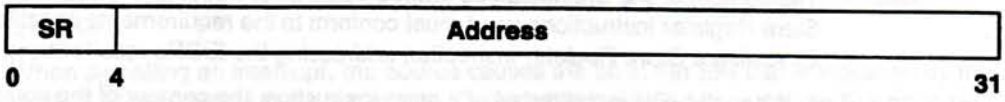


Figure 18. Effective Address

SR These four bits must select the Segment register set up to access the architectural resources of BUID 0.

Address This field contains the address of the EICRs being accessed.

The address assignments for the EICR are shown in the following list:

Register	Hex Address
EIM0	0000000
EIM1	0000004
Reserved	0000008
Reserved	000000C
EIS0	0000010
EIS1	0000014
Reserved	0000018
Reserved	000001C
DEC EISBID	0000020.

Accessing the EICRs

The EICRs can be accessed using load or store instructions that conform to the following requirements:

- All accesses must be on a word boundary.
- Data transfers must be multiples of four bytes (one word).
- Multiple word transfers are not interruptible.
- When the EIS is the target of a load or store instruction and any external interrupt requests are signalled to the EIS during the execution of that instruction, the result is the same as if the events occurred sequentially. The signalled interrupts are not lost or duplicated.
- Accesses that address reserved locations either directly or through multiple word transfers cause a data storage interrupt when the reserved location is addressed. Registers or locations altered before the interrupt are not restored.

Reading from the EICRs

The content of the EICRs can be copied to one or more GPRs using load instructions addressing the registers. The instruction used must conform to requirements previously specified.

- The content of the addressed registers replaces the content of the target registers.
- When the EIM is the source for the load, the content of the EIM is not altered.
- When the EIS is the source for the load, the content of the addressed portion of the EIS is set to zero.

Writing to the EICRs

The content of the EICRs can be altered using a store instruction addressing them. The Store Register instructions used must conform to the requirements specified previously. Executing a Store Register instruction addressing the EICRs results in the following actions:

- When the EIM is the target of a store instruction, the content of the source registers replaces the content of the specified portion of the EIM.
- When the EIS is the target of a store instruction, the content of the source registers is ORed with the content of the specified portion of the EIS (one or two words) and the result replaces that portion of the EIS.
- The change may affect the EIR or the EIRS, or both.

Notes:

1. Instructions addressing the EICRs that transfer more than 4 bytes are *not* interruptible.
2. The bits of the EIS can be mapped to hardware interrupt sources. However, any of the EIS bits can be set by software.

External Interrupt Sources

The description is necessary to define the interface between the processor and the I/O process.

An External Interrupt source is a logical entity that is associated with a specific bit in the EICRs. Whenever a source recognizes the need to be serviced by the processor, it submits a request to have that bit set to 1. The association of bits in the EICRs and interrupt sources is programmable and set by software as needed.

Submitting Interrupts

Submission of interrupt requests to set bits in the EIS must conform to the following requirements:

- The source must not lose any interrupt requests.
- The source should minimize the redundant submission of interrupt requests for any single event that requires servicing by the processor.
- The source should not submit any requests if it can determine that a previous request it submitted is still pending.

Note: This function may require the implementation of a latch that is set when a request is first submitted and must be reset by software after the interrupt is serviced.

EICR Mapping

An interrupt source sets the bit of the EIS to which it has been programmed to by software. This feature requires additional functions of the interrupt source.

- A source contains a location (EISBID) that can be read and altered as desired by software.
- Transfer EISBID Content to GPR: A load instruction addressing any EISBID must transfer at least one word. EISBID is placed in bits 26 to 31 of register RT. Bits 0 to 25 of register RT are undefined.

- Transfer GPR to EISBID: A store instruction addressing any EISBID must transfer at least one word. The contents of bits 26 to 31 of register RS are placed in the addressed EISBID.
- When signalling an interrupt, the source causes the bit in the EIS that is indicated by the content of the EISBID to be set to one. Only six bits are used to select the EIS bit to be set.

Note: The Decrementer (DEC) causes an External Interrupt that is associated with a bit in the EICRs. The EISBID for the Decrementer is located in the I/O space as shown in "Addressing the EICRs" on page 1-64.

External Interrupt Mechanism for POWER2

This interrupt mechanism provides a means for sensing, presenting, and controlling interrupts. All interrupts are classified by level. The interrupts are presented to the processor in order of most favored interrupt first.

The External Interrupt mechanism is composed of the Interrupt Level Control register and associated hardware that provides the means by which software can manage External Interrupts. The Interrupt Level Control register (accessible as a Special Purpose register) and the External Interrupt mechanism provide a means for software to perform the following tasks:

- Sense the current interrupt level.
- Sense the pending interrupt level.
- Clear a pending interrupt at any selected level.
- Set an interrupt at any selected level.

An interrupt level can be one of 64 levels (0 to 63). Level 0 is the most favored level and level 63 is the least favored level. The current interrupt level (CIL) can be set to any value between 0 and 255. A processor can accept interrupts only when pending interrupt level is more favored than the level indicated by ILCR(CIL).

For example, if the content of ILCR(CIL) = 37, the processor accepts interrupt levels 0 through 36. Interrupt levels 37 through 63 are masked.

When an interrupt is signalled, it is posted and remains pending until software dispatches or resets the interrupt.

Interrupt Level Control Registers

The Interrupt Level Control register (ILCR) is a 32-bit register that provides the interface through which software manages the External Interrupt mechanism. See Figure 19.

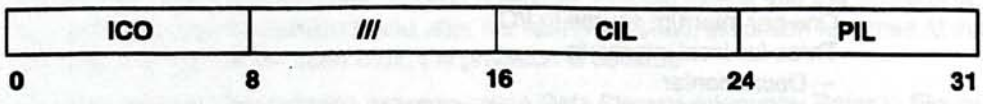


Figure 19. Interrupt Level Control Register

The ILCR contains three 8-bit fields (ICO, PIL, CIL). The content and function of these fields depend on the instruction executed.

The register is manipulated with the mfspr and mtspr instructions. The following sections describe the result of executing those instructions.

MFSPR RT, ILCR

This instruction copies the content of the ILCR into the RT register.

- CIL contains the interrupt level at which the processor is executing. The processor accepts only an interrupt that is at a more favored level than the level indicated by the CIL.
- PIL contains 255 if no interrupt is pending. Otherwise, it contains the level of the most favored pending interrupt. The returned interrupt level is removed from the list of pending interrupt levels.
- Bits 0 to 15 of the RT register are set to x'0000.'

MTSPR ILCR, RS

This instruction causes the interrupt control mechanism to execute the command contained in bits 0 to 7 of the RS register.

ICO	Interrupt Mechanism Action
00000000	Update CIL (UCIL) Copies the content of bits 16 to 23 of the RS register to the CIL field of the ILCR.
00000001	Clear Interrupt Level (CIL) Clears the interrupt level specified by bits 24 to 31 of the RS register.
00000010	Set Interrupt Level (SIL) Sets the interrupt level specified by bits 24 to 31 of the RS register.

All other values of the ICO field are ignored.

Notes:

1. An mfilcr instruction closely following an mtilcr instruction does not obtain the actual values of the ILCR register.
2. External interrupts need not be disabled to write to ILCR.

EISBID Registers

A 6-bit External Interrupt Status Bit ID register (EISBID) is associated with each interrupt source. Software can write the EISBID register to assign the priority at which that source will signal that an interrupt is pending. In addition to external sources, each of the three local interrupt sources also have an associated EISBID.

- One per interrupt source in I/O
- Three for local interrupts
  - Decrementer
  - External check (memory error on DMA)
  - Early power-off warning (EPOW).

The following are the EISBID address assignments in BUID 0:

Register	EISBID Address
Decrementer	0000020
External Check	0000024
EPOW	0000034

PEIS Registers

The pending External Interrupt Status registers contain pending interrupt status for the 64 interrupt levels. Using the addresses in BUID 0, software can read these registers.

The following are the address assignments in BUID 0:

Register	Address
PEIS0	X'0000 0010'
PEIS1	X'0000 0014'

Storage Control

This section describes the function of and control over the storage mechanism. Brief motivation is given, but the primary purpose of this section is to serve as a reference. Some of the major features of the storage mechanism are as follows:

- Page size is 2^{12} bytes.
- Maximum real memory size is 2^{32} bytes.
- Presumed minimum real memory size is 2^{24} bytes.
- Virtual memory size is 2^{52} bytes.
- Number of segments is 2^{24}.
- Number of transaction IDs is 2^{16}.
- Hardware support for Special Segments (physical lock management on a 128-byte line).
- Automatic granting of locks in Special Segments in some cases.
- Memory-mapped I/O into I/O segments.
- IPL ROM origin at address X'FFF0 0000'.

The memory hierarchy of the system consists of the following two levels:

- Cache
  - Instruction cache
  - Data cache that is managed store-in.
- Main memory.

Instructions are provided to manage a data cache and an instruction cache. The I/O goes directly into main memory with no hardware interrogation of the caches. Software must issue the necessary cache control instructions before issuing an I/O to ensure consistency of the data cache, instruction cache, and main memory. Instructions can be changed by treating them as data in the normal way. A store to the data cache is not guaranteed to update the instruction cache. Again, software must issue the necessary cache control instructions to maintain the consistency of the two caches, instruction prefetch and main memory.

Page faults cause precise Data Storage Interrupts. Precise means that the address of the faulting instruction is identified, and after the fault is satisfied, execution resumes at that address. For instruction page fault, the precision is obvious.

Crossing segment boundaries can also cause Data Storage Interrupts. Refer to Figure 28 on page 1-77 for more information.

For Data Storage Interrupts, the precision is present but there may be side effects. In general, an instruction that makes a reference that causes a Data Storage Interrupt does not change the contents of any register that can be changed in nonprivileged state, which would prohibit restarting the instruction after the interrupt is serviced by software.

In those cases where registers or storage are changed, they are not changed in a way that would prevent the restart of the faulting instruction. Examples of such instructions are Load Multiple (lm), Load String Indexed (lsx), Load String and Compare Indexed (lscbx), Load String Immediate (lsi), Store Multiple (stm), Store String Indexed (stsx), and Store String Immediate (stsi). The lm, lsx, lsi, or lscbx instruction may fault part-way through its execution with only some of the specified registers actually loaded. The lm, lsx, lsi, and lscbx instructions are restartable since the base registers are not altered, even if they are in the range to be loaded. The stm, stsx, and stsi instructions may also fault part-way through. In this case, some of the storage locations destined to hold the registers being stored may have changed as well as respective page table entries. However, stm, stsx, and stsi are restartable since the base registers are not altered. Unaligned stores may update storage prior to the fault and leave the job up to the relevant interrupt handler to complete.

Crossing page or segment boundaries by a single instruction is not necessarily prohibited by this architecture. However, each side of the boundary must adhere to the specific rules for protecting that side. Crossing a segment boundary, however, results in a Data Storage Interrupt if the first segment accessed has T = 0 and the second segment has T = 1. Crossing a segment boundary cannot occur when the first segment has T = 1. The Segment register of the calculated effective address indicates T = 1, and the processor sends the load/store command with the Segment register and effective address to I/O with no further checking of the Segment registers.

Storage Control Registers

The following section describes the Segment registers and the Storage Description registers.

Segment Registers

There are sixteen 32-bit Segment registers (SR) as shown in Figure 20. The most significant bit of a Segment register is called the T bit. When T = 0, the segment named in the Segment register is a normal segment. When T = 1, as shown in Figure 22, the segment named in the Segment register is an I/O segment. Unless explicitly noted, all discussions of segments from this point on deal only with normal segments.

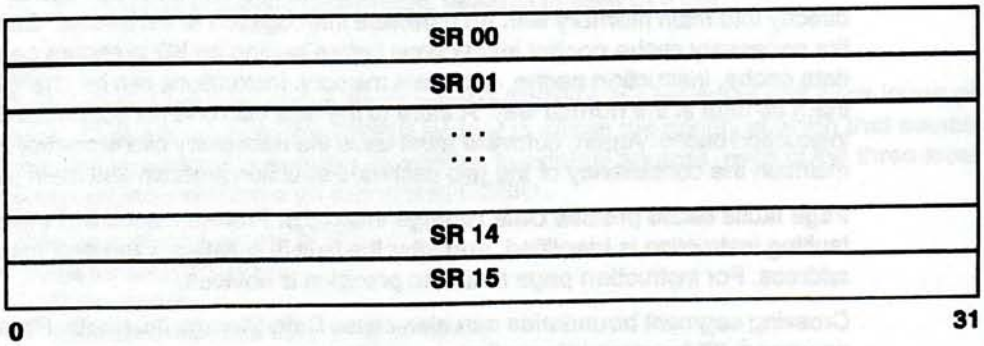


Figure 20. Segment Registers

Segment registers, when T = 0, contain a 24-bit Segment ID (SID), a Special Segment (S) bit, and a 1-bit segment access key (K), in the format presented in Figure 21.

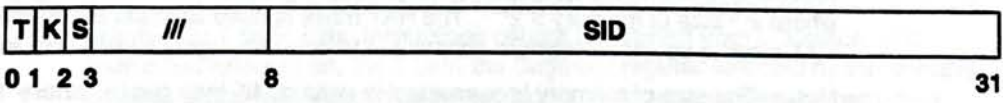


Figure 21. Segment Register Format (T = 0)

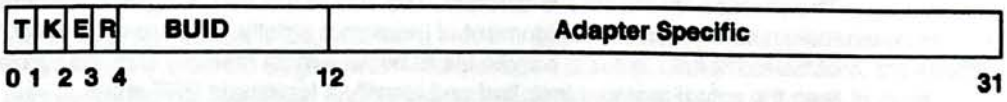
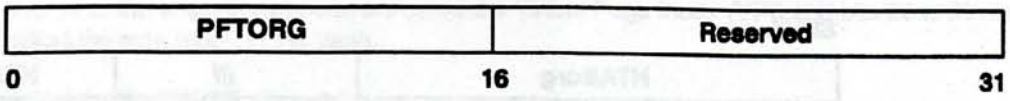


Figure 22. Segment Register Format (T = 1)

Storage Description Registers for POWER

The Storage Description registers (SDR0 / SDR1) shown in Figure 23 are 32-bit registers. SDR0 contains the high-order bits of the real address of the Page Frame Table (PFT). SDR1 contains the high order-bits of the real address of the Hash Anchor Table (HAT) and the HAT mask. Access to these registers by software is privileged.

SDR0



SDR1

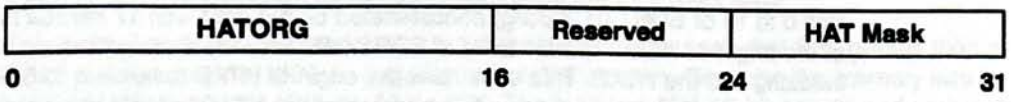


Figure 23. Storage Description Registers

Bits 0 to 15 of SDR0 (PFTORG) concatenated on the right with 16 zeros is the real address of the origin of PFT. Bits 0 to 15 of SDR1 (HATORG) concatenated on the right with 15 zeroes is the real address of the origin of HAT. Bits 24 to 31 of SDR1 (HAT mask) to contain the mask to be used when indexing into the HAT. This constrains the origin of HAT to be on a 32K-byte boundary and the origin of the PFT to be on a 64K-byte boundary. These alignment constraints permit the relocation hardware to index into the tables without addition when the machine implements the smallest presumed main memory size (16M byte).

The reason this limit is presumed is that the actual amount of real memory may be less, but the hardware addressing through the HAT assumes the limit. If less memory than the presumed minimum is actually installed, then software cannot use any PFT entry that corresponds to the noninstalled page of real memory as a legitimate entry without causing an error. As the size of main memory increases, the number of bits used to index the HAT increases. Thus software must adjust the base address of the HAT such that this address has at least the same number of low-order zeros as the hardware has additional bits of index.

This implies a hardware variable merger based on actual real memory size. The number of entries in the HAT is a trade-off between HAT size and the average PFT chain length. It is recommended that the number of entries in the HAT = $2^r + 1$,
where $2^{r \geq}$ size of memory > 2^{r-1}. The HAT mask is used to assist the hardware merger. The HAT mask is set by software to contain r – 12 one bits, right-justified with leading zero bits.

Note: The size of memory is expressed in units of 4K-byte pages, where $12 \leq r \leq 20$ (presumed MIN and absolute MAX number of 4K-byte pages).

The situation for the PFT is different. The size of the PFT is again determined by the size of main memory, but the assignment of indexes is strictly software-controlled. While the width of the index field is set to handle the maximum main memory size, no value can be bigger than the actual memory installed and identify a legitimate PFT entry.

For the PFT, hardware ORs the index with the base independent of the installed memory size. Software must adjust the base of the PFT (PFTORG) so that there are sufficient low-order zeros not to conflict with the maximum index value for the installed size.

Storage Description Register for POWER2

The Storage Description register 1 (SDR1) is a 32-bit registers (shown in Figure 24). SDR1 contains the high-order bits of the origin address of the hashed page table (HTAB) and the HTABmask. Access to these registers by software is privileged.

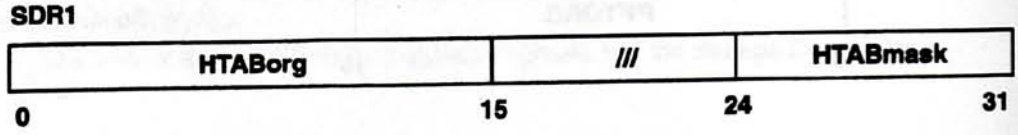


Figure 24. Storage Description Register

Bits 0 to 14 of SDR1 (HTABorg) concatenated on the right with 17 zeroes is the real address of the origin of HTAB. Bits 24 to 31 of SDR1 (HTABmask) contain the mask used when indexing into the HTAB. This constrains the origin of HTAB to be on a 128K-byte boundary. Thus software must adjust the origin address of the HTAB to have at least the same number of low-order zeros as the hardware has additional bits of index.

This implies a hardware variable merge based on actual real memory size. The HTABmask is used to assist the hardware merge. The HTABmask is set by software to contain r – 12 one bits, right-justified with leading zero bits.

Assume r is the smallest integer such that 2^r is greater than or equal to the number of 4K-byte page frames of real memory. Since the memory size is between 16M bytes and 4G bytes, $12 \leq r \leq 20$.

Virtual Address Translation

Translation is enabled by 2 bits in the MSR; there is one bit for data address translation MSR(DR) and one bit for instruction address translation MSR(IR). MSR(IR) and MSR(DR) are independent bits and can be set differently. These bits are changed by the mtmsr instruction which must be executed in privileged mode. Changing either of these bits with mtmsr is synchronizing. That is, fetching and executing the instructions after the mtmsr instruction is performed according to the new settings of the MSR. The new settings of the MSR may not immediately affect the fetching of instructions because an instruction beyond the mtmsr instruction may have been fetched prior to the execution to the mtmsr. ICS can be

used to discard the prefetched instructions and begin fetching instructions following the ICS in the mode specified by the new setting of the MSR.

Note: Accessing of I/O is independent of MSR(DR) because access to I/O is controlled only by the T bit in SRs. Instructions cannot be fetched from I/O space. With instruction-relocate on, the T bit in the Segment register selected by the effective address of the next sequential instruction must be zero, or else an Instruction Storage Interrupt is generated. When MSR(IR) = 0 the T bit is ignored for instruction fetches.

When data translation is off, MSR(DR) = 0, the Segment register is only accessed to determine if it is an I/O segment for data storage accesses. If the T bit is zero, the effective address is the real address, and its numerical value is the address of a byte in main memory. If the T bit is one, the effective address is sent to I/O.

When address translation is enabled, the hardware supports a 52-bit single virtual address space consisting of up to 2^{24} segments of 256M bytes each in 4K-byte pages. This address is formed by the processor generating a 32-bit effective address that refers either to an instruction or to data.

The translation hardware has 16 Segment registers. Bits 0 to 3 of the effective address are used to address a Segment register. The 24-bit SID field of the accessed Segment register is concatenated with bits 4 to 31 of the effective address to form a 52-bit virtual address. Bits 4 to 19 of the effective address are called the Virtual Page Index (VPI) and bits 20 to 31 are called the byte offset in the page.

Inverted Page Table (POWER Only)

The address tables that define the mapping from virtual to real addresses are comprised of the Hash Anchor Table (HAT) and Page Frame Table (PFT). These tables are maintained by software and are searched by the relocation hardware as a hash table.

Remember, as described previously, r is the smallest integer such that 2^r is greater than or equal to the number of 4K-byte page frames of real memory. Because the memory size is between 16M byte and 4G byte, $12 \le r \le 20$. The HAT has 2^{r+1} 32-bit entries and each entry in the HAT contains an index into the PFT with an invalid bit, i. See Figure 25.

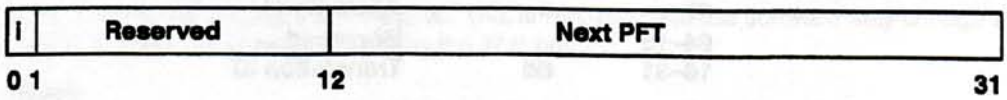
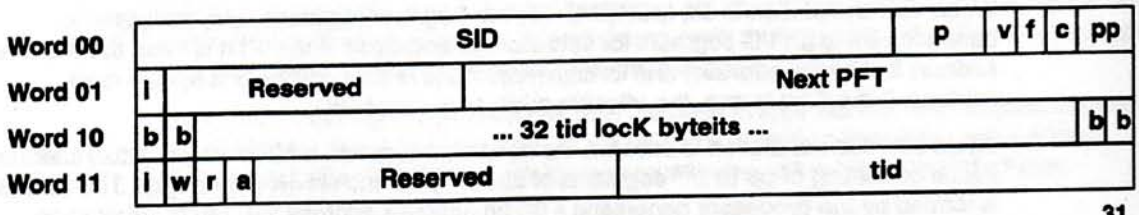


Figure 25. Hash Anchor Table Entry (One Word)

The number of entries in the PFT is 2^r; for example, one per real page frame of memory in the case when the size of real memory is a power of 2. Each entry in the PFT spans one quadword (four words) and has the format shown in Figure 26.

Note: Bits 1 to 11 of each HAT entry, bits 1 to 11 of word 1, and bits 4 to 15 of word 3 of each PFT entry should not be used by software and should be treated as reserved for future use. Some of these bits may be inadvertently altered by hardware when updating PFT information.



	Bits	Symbol	Description
Word 00	00–23	SID	Segment ID
	24–26	p	3 high-order bits of VPI
	27	v	SID valid
	28	f	Reference bit
	29	c	Change bit
	30–31	pp	Page Protection Keys
Word 01	00	l	Invalid bit
	01–11		Reserved
	12–31	Next PFT	Frame number of next PFT entry
Word 10	00–31	b	Lock byteits for lines
Word 11	00	l	Locktype
	01	w	Grant Write Locks
	02	r	Grant Read Locks
	03	a	Allow Read
	04–15		Reserved
	16–31	tid	Translation ID

Figure 26. Page Frame Table Entry (One Quadword)

To translate a virtual address to real the HAT and PFT are searched by the relocation hardware as follows.

Bits 0 to 3 of the effective address are used to select a Segment register. The selected Segment register contains the 24-bit SID. Eight zeros concatenated with the VPI (bits 4 to 19 of the effective address) are XORed with the 24-bit SID. The low order r + 1 bits (specified by the HAT mask) of the result are used as an index into HAT. The real address formed to look up the entry in the HAT is as follows:

```
opa = b'0' || HATORG || 15 x b'0'
opb1 = (X'00' || VPI) xor SID
opb2 = b'000' || HAT mask || 13 x b'1'
opb3 = opb1 and opb2
opb = 6 x b'0' || opb3 || b'00'
real address = opa or opb
```

If the next pft field of the selected entry in the HAT is invalid (i = 1), then the search fails. Otherwise, the next pft field is an index used to select an entry from the PFT. The real address formed to select an entry in the PFT is:

```
opa = PFTORG || 16 x b'0'
opb = X'00' || next pft || X'0'
real = opa or opb
```

where *next pft* comes initially from the HAT, and subsequently from the PFT.

Having selected an entry in the PFT, SIDII(bits 0 to 2 of the VPI) are compared to bits 0 to 26 of word 0 of that entry. If they compare equally and the SID valid bit (V) is a one, the search succeeds.

Otherwise, the search continues by accessing the next PFT entry indexed by *next pft* (bits 12 to 31 of word 1 of the current PFT entry) if the Invalid bit (bit 0 of word 1 of the current PFT entry) is 0, and repeating the process. If the Invalid bit is 1 then the search fails.

Note: All hardware lookups are done through the cache using real addressing.

To prevent an infinite loop in this search, the hardware searches for a maximum of 127 entries during a single translation. If this limit is exceeded, the search fails.

When the search succeeds, the real page frame is the index of the PFT entry that contained the matching virtual address, and the real address is obtained by concatenating that index with bits 20 to 31 of the effective address. When the search fails there is no real address associated with the virtual address, and a Data/Instruction Storage Interrupt is generated.

The translation between virtual and real addresses is defined by the HAT and PFT, and conceptually these tables are searched by the address relocation hardware to translate every reference. However, for performance reasons the hardware keeps a Translation Look-aside Buffer (TLB) that holds portions of the PFT that it has recently used, and the TLB is searched before referring to the tables in storage. As a consequence, when software makes changes to these tables, it must issue the appropriate TLB purge instructions to maintain the consistency of the TLB and the tables.

When a TLB entry is loaded, hardware must insure that either all of the lock and TID information is loaded from the PFT entry, or that the information is marked invalid in the TLB entry, even if the segment is not special. This is required because software may change the S bit of a segment without invalidating the TLB entry.

Notes:

1. It is possible for the hardware to actually implement two sets of SRs (one for data and one for instruction). In this case, hardware must insure that the same numbered register in both sets have the same value. Likewise, it is possible for the hardware to implement two TLBs. In this case, the size, shape, and values contained may be different, but the hardware must invalidate both TLBs as part of the execution of a single TLB instruction.
2. If floating stores are used to update PFT entries, a Data Cache Synchronize (dcs) should be used to insure the store operation completes.

The physical location (in cache) of data or instructions is governed by bits 12 to 31 of the effective address regardless of whether the address translation is enabled or disabled. For this reason, care must be exercised by software in referencing data with translate off which have been previously referenced with translate on, and the oposite situation. Let ra denote the real address corresponding to an effective address after translation. Since only bits 20 to 31 are unaffected by translation, bits 12 to 19 of ra and ea may differ. When they do differ, an effective address of ra with translate off refers to a different physical location in cache than ea with translate on, potentially leading to inconsistent results. This can be avoided by

software restricting the data that are referenced with translate on and off to have bits 12 to 31 of the real and virtual addresses agree, or by the appropriate use of the Flush/Invalidate Cache instructions.

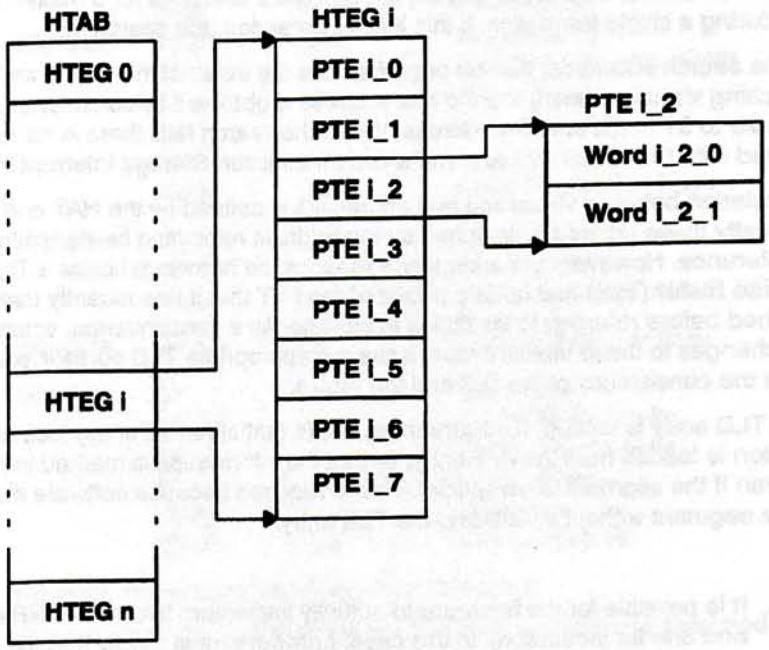
Note: The HAT and PFT are accessed by hardware using real addresses. Care must be taken if software accesses these with translation on.

If a storage access has the effect of updating the corresponding PFT entry, then that PFT entry is updated prior to the next storage access.

Hashed Page Table (POWER2 Only)

The HTAB contains a maximum of 2^{19} Hash Table Entry Groups (HTEGs). Fewer HTEGs can be allocated if software sets HTABmask appropriately. For a system with N pages of real memory installed, the proposed number of HTEGs that should be allocated is N/2. See Figure 27.

The HTAB must be located in a contiguous block of storage and cannot contain any defective areas.



Notes:

1. **n = (N/2) – 1 where N is the number of real pages of storage.**
2. **HTEG(i) is one of N/2 hash table entry groups.**
3. **PTE(i_j) is one of 8 page table entries in HTEG(i)**

Figure 27. Virtual Address Translation Data Structure

An HTEG contains eight page table entries (PTEs). HTEGs are the addressable element in the HTAB. Hashing the virtual address, described in the following discussion, produces a pointer to the first of two HTEGs that could contain the translation for the virtual address, if a translation exists. If the translation is not found in the initial HTEG, the virtual address is rehashed and a secondary HTEG is searched.

Each two-word PTE contains fields to specify the Segment ID (SID), the abbreviated virtual page index, the real address, page protection, the reference bit, and the change bit. The contents of the PTEs are shown in Figure 28.

The two words in the PTE describe the real page. The 20-bit RPN field specifies the real page number (RPN) of the page translated by Word 0 of this entry.

Note: Reserved bits in the PTE should not be used by software. Some of these bits can be inadvertently altered by hardware when updating information.

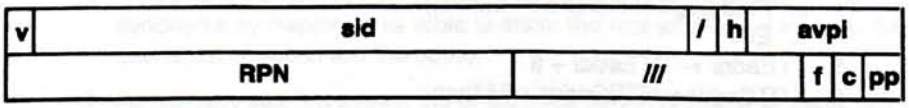


Figure 28. Page Table Entry (Two Words)

Word	Bit	Symbol	Description
0	0	v	entry valid
	1–24	sid	segment id
	25		reserved
	26	h	hash function selector
	27–31	avpi	abbreviated vpi, EA(4–8) = VPI(0–4)
1	0–19	RPN	real page number
	20–27		reserved
	28	f	reference bit
	29	c	change bit
	30–31	pp	page protection keys

Hashed Page Table Search

To translate a virtual address to a real address, the relocation hardware searches the HTAB as follows: bits 0 to 3 of the effective address are used to select a Segment register. The 24-bit SID is extracted from bits 8 through 31 of the selected Segment register. Then bits 4 through 19 of the effective address (VPI), the 24-bit SID, the HTABorg, and the HTABmask are used to select one of N/2 HTEGs within the HTAB (where N is the number of pages of configured memory). Software sets the size of the HTAB by setting the HTABmask in SDR1. The real address (HTEGaddr), formed to access the initial HTEG, is shown in Figure 29.

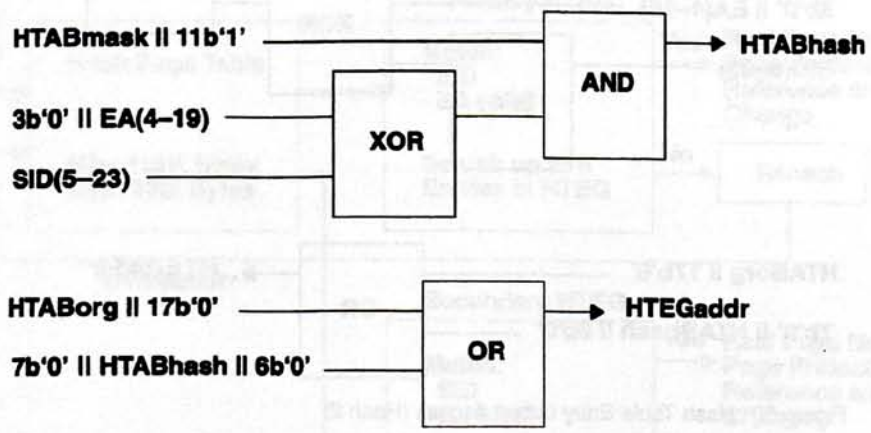


Figure 29. Hash Table Entry Group Access

If a second hash is required (see step 8), the real address (HTEGaddr') formed to access the second HTEG is shown in Figure 30. The initial and secondary HTEGs are searched for the missing translation as follows:

1. PTEaddr ← HTEGaddr
2. Access PTE at memory location PTEaddr
3. If PTE(v) = 0 or PTE(h) = 1, then go to step 5
4. If SID || EA(4–8) = PTE(sid || avpi) then
   a. Translation succeeds
   b. Real page number ≤ PTE(RPN)
   c. Exit
5. PTEaddr ← PTEaddr + 8
6. If PTEaddr = HTEGaddr + 64 then
   a. All eight entries are searched; no match is found
   b. Rehash virtual address (see Figure 30)
   c. Go to step 8
7. Go to step 2
8. PTEaddr ← HTEGaddr'
9. Access PTE at memory location PTEaddr
10. If PTE(v) = 0 or PTE(h) = 0 then go to step 12
11. If SID || EA[4–8] = PTE(sid || avpi) then
    a. Translation succeeds
    b. Real page number ≤ PTE(RPN)
    c. Exit
12. PTEaddr ← PTEaddr + 8
13. If PTEaddr = HTEGaddr + 64 then
    a. All eight entries are searched; no match is found
    b. Translation fails
    c. Generate Data/Instruction Storage interrupt
    d. Exit
14. Go to step 9

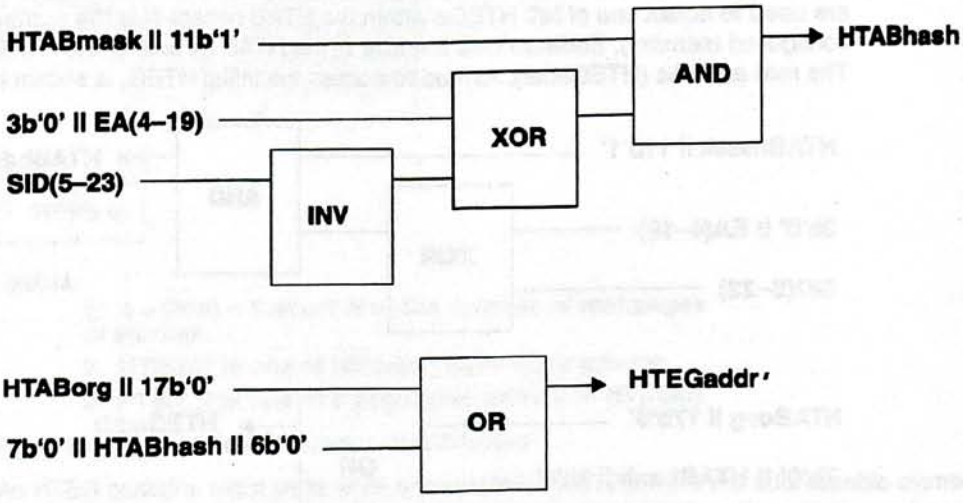


Figure 30. Hash Table Entry Group Access (Hash 2)

All eight PTEs in the primary HTEG and all eight PTEs in the secondary HTEG are checked until a matching entry is found. If no matching PTE is found in either HTEG, the translation fails and a Data/Instruction Storage interrupt is generated.

Notes:

1. HTAB entries may or may not be cached by hardware.
2. All hardware lookups are done through the cache using real addressing.
3. If software will access the HTAB in translated mode, it must avoid cache line synonyms by mapping this table to make the real and virtual address bits used for cache set selection are the same.
4. As memory size increases, the origin must be set to make as many low-order bits of the origin zero as there are significant bits on in the offset.
5. SDR1: HTABorg and HTABhash must be valid for the installed memory size. HTAB contains:
   - Minimum: 2^{11} HTEGs $\rightarrow$ 2^{14} PTEs$\rightarrow$ 2^{12} real pages
   - Maximum: 2^{19} HTEGs$\rightarrow$ 2^{22} PTEs $\rightarrow$ 2^{20} real pages

When the search succeeds, the real page number is the RPN in the selected page table entry that contained the matching virtual address, and the real address is obtained by concatenating that RPN with bits 20 to 31 of the effective address.

The translation between virtual and real addresses is defined by the HTAB. Conceptually, this table are searched by the address relocation hardware to translate every reference. However, for performance reasons the hardware keeps a translation look-aside buffer (TLB) that holds recently used PTEs, and the TLB is searched before referring to the table in storage. As a consequence when software makes changes to this table it must issue the appropriate TLB invalidate instructions to maintain the consistency of the TLB and the table. See Figure 31.

Note: If floating-point stores are used to update HTAB entries, a data cache synchronize (DCS) should be used to insure the store operation completes.

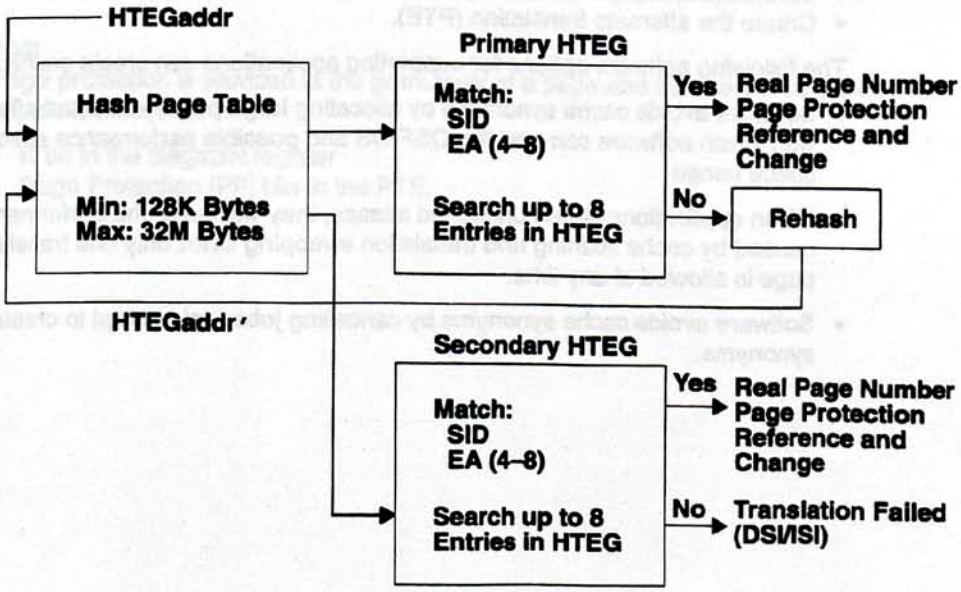


Figure 31. Virtual Address Translation

Address Aliasing

In multiple tasking systems, aliasing as used in reference to the virtual address mechanism means the concurrent use of multiple virtual addresses to access a single storage element. This architecture supports aliasing. To maintain storage consistency in this environment, software must obey the following rules:

- Write Shared Data

  The aliases used to access write shared data must be aligned on 256K-byte boundaries (bits 14 through 31 of the addresses used to access the data must be identical).

- Read Shared Data

  The allowed synonyms used to access read shared data (read only data or instructions) must be aligned on 4K-byte boundaries.

- Data Shared with I/O

  The I/O architecture defines a storage model that is not consistent with I/O. Cache consistency must be managed by software using the cache management instructions to insure that changes to storage caused by I/O operations are visible to the processor, and those caused by the processor are visible to I/O.

These restrictions are necessary to avoid the creation of synonyms that could cause storage to be inconsistent. A synonym is created when the main memory copy of a storage element residing in one set of the cache is copied from main memory into a different set of the cache because a second virtual address was used for a subsequent access and bits 16 through 31 of the addresses differ. The creation of synonyms when accessing read only data is not a problem unless that storage must be consistent with I/O.

When a storage element is addressed using multiple addresses but accesses using different addresses are not concurrent, aliasing can be avoided. This approach is useful only in cases where the storage is accessed by one address for a long period and then by a different address for a long period as in the case of cooperating processes that are not allowed to run concurrently. When a change of address spaces occurs:

- Purge the cache of the shared address space
- Delete the existing translation (PTE)
- Create the alternate translation (PTE).

The following software options for supporting applications can create cache synonyms:

- Software avoids cache synonyms by allocating large pages. This had effects on the ease with which software can port the OSF OS and possible performance effects on address space usage.
- When applications create unaligned aliases, they will suffer the performance penalties caused by cache flushing and translation swapping since only one translation for the page is allowed at any time.
- Software avoids cache synonyms by cancelling jobs that attempt to create cache synonyms.

A similar effect results when storage is accessed in both real and translated modes. In this case there are two aliases, the real address and the virtual address. The same rules hold for these accesses. The addresses' spaces must be aligned or the data must be purged from the cache between accesses using one address and accesses using the other.

Storage Access Recording Mechanism

Reference and Change bits are maintained in the PTE, if address translation is enabled, for each real page, and can be accessed by software directly through ordinary load and store instructions. These bits are set automatically by hardware in conjunction with normal TLB processing as follows:

Reference bit When a storage access (load, store, or cache instruction, or instruction fetch) results in a TLB miss and the resulting translation is loaded into the TLB, the reference bit may be set to 1 immediately, or its setting may be delayed until the storage access is determined to be successful. If the reference bit is not set because the access failed, the implementation must set the reference bit on the next successful access.

Change bit Whenever a data store is executed, as part of the TLB look-up procedure, the change bit in the TLB is checked and if it is already set to 1, no further action is taken. However, if the TLB change bit is 0, it is set to 1, and the corresponding change bit in the PTE is set to 1.

Note: Since hardware only sets the Reference and Change bits on the basis of TLB activity, when software resets these bits to zero, it must synchronize the TLBs actions by invalidating the TLB entries associated with the pages whose reference and change bits are reset.

Also, since some implementations may not set the Reference bit when a TLB entry is loaded due to an unsuccessful storage access, this indicates there may exist an entry in the TLB for the page even though the reference bit in the PTE is 0.

Storage Protection Mechanism

The protection mechanism is provided to protect the contents of main storage from destruction or misuse caused by unauthorized accesses by a program.

Page Protection

Page protection is provided at the granularity of a page and the mechanism uses two separate fields:

- K bit in the Segment register
- Page Protection (PP) bits in the PTE.

Storage protection applies only when address translation is enabled. A reference made with translation enabled is associated with a Segment register (SR) and a PTE by the address translation procedure described in the preceding section. The following table describes the access permitted in terms of the value of the access key in the Segment register and Page Protection bits in the PTE.

Protection Key Processing				
K	PP	Page Type	Load Access Permitted	Store Access Permitted
0	00	Read and Write	Yes	Yes
0	01	Read and Write	Yes	Yes
0	10	Read and Write	Yes	Yes
0	11	Read only	Yes	No
1	00	No access	No	No
1	01	Read only	Yes	No
1	10	Read and Write	Yes	Yes
1	11	Read only	Yes	No

Where:

K = Segment register access key
PP = PTE page protect bits.

When a reference is not permitted because of the protection mechanism, a Data Storage Interrupt (Instruction Storage Interrupt) occurs and bit 4 of the DSISR (SRR 1) is set to 1.

The I/O protection mechanism provided in the case of an I/O segment, the K bit, provided with the effective address, is used to protect I/O facilities.

Timer Facilities

The *real-time clock* (RTC) and the *decrementer* (DEC) provide the timing functions for the system. Both functions are manipulated as Special Purpose registers.

Real Time Clock The RTC provides a high-resolution measure of real-time suitable for the indication of date and time of day. This is a volatile resource and must be initialized during start-up.

Decrementer The decrementer provides a means of signalling an interrupt after a specified amount of time has elapsed unless the decrementer is altered in the interim.

Real-Time Clock

Note: This architecture provides no functions to synchronize clocks in a cluster.

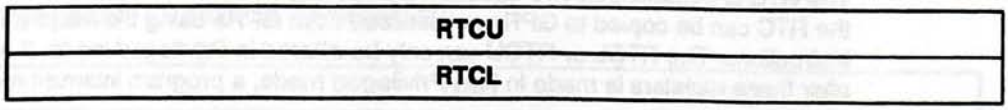
The real-time clock is composed of two Special Purpose registers as shown in Figure 32. RTCU is the count of seconds since the epoch specified by software architecture. RTCL is a measure of the fraction of the current second in nanoseconds such that when used with

RTCU it provides a high-resolution measurement of the real time. The RTC provides a calendar range of 136.19 years. The following requirements apply to both registers:

- The RTC runs continuously when powered on, but has no provision for retaining the correct time during power-off periods.
- On power up, the RTC begins running but the content is undefined.
- The RTC can be used to measure elapsed time prior to initialization by first setting RTCL to 0 and then comparing the values read from the RTC at times bracketing the period of interest.

Notes:

1. Software must initialize the RTC from a time source external to the processor.
2. If RTCL is not initialized, or if it is initialized with a value greater than 999,999,999, the elapsed time until the count is reset to 0 by hardware is undefined but less than four seconds.



Register	Description
RTCU	Represents time in seconds.
RTCL	Represents time in nanoseconds.

Figure 32. Real Time Clock (RTC)

Note: All bits in the RTCL need not be implemented.

RTCL Description

All 32 bits of RTCL need not be implemented. The driving frequency, insertion bit, controls, and number of bits must be implemented such that the following requirements are satisfied:

- If all bits are implemented, bit 31 of RTCL changes state each nanosecond.
- The implemented bits function as a binary counter.
- The initial implementation has a resolution of 256 nanoseconds.
- The period of RTCL is one billion nanoseconds (one second) once the content is set to 0 by software or hardware. This occurs within four seconds of power up if not initialized by software.
- When not being altered by software, the RTCL operates such that:
  - Only 999,999,999 nanoseconds after it has been set to 0, the content is equal to the terminal count (a value less than and as near to 999,999,999 as the implemented bits allow).
  - One nanosecond later the content is set to 0.
- Moving the content of RTCL to a GPR has no effect on the counter. After the move, bits in the GPR corresponding to the unimplemented bits in the counter are 0's.
- Moving the content of a GPR to the counter causes the contents of the implemented bits of RTCL to be replaced by the contents of the associated bits of the source GPR. Bits in the GPR corresponding to the unimplemented bits in RTCL are ignored.

RTCU Description

RTCU is a 32-bit binary counter which satisfies the following requirements:

- When the next state of RTCL is to become 0 because it has reached terminal count, RTCU is incremented in synchronism with the setting of RTCL to 0.
- All 32 bits of RTCU are implemented.
- When the content of RTCU is all 1s, the next time it is incremented the content becomes all 0s.
- The counter runs continuously while powered on.
- Moving the content of RTCU to a GPR has no effect on the counter.
- Moving the content of a GPR to RTCU causes the content of RTCU to be replaced by the content of the source GPR.

Setting and Reading the RTC

The RTC is accessed as two Special Purpose registers, RTCU and RTCL. The contents of the RTC can be copied to GPRs or initialized from GPRs using the mtspr and mfspr instructions. The RTCL or RTCU can only be altered in Privileged mode. If an attempt to alter these registers is made in Non-Privileged mode, a program interrupt results.

Initializing the RTC

The content of the RTC can be altered using the mtspr instruction. This is a privileged access.

The RTC can be initialized by the following sequence of instructions and commands:

1. Load the value X'0000 FFFF' into Rx.
2. Obtain the correct time from a source external to the processor.
3. Compute a 32-bit representation of this value in seconds and place in Ry.
4. Compute the residual fractions of a second in nanoseconds and place in Rz.
5. Issue mtspr RTCL,Rx. Set lower register to zero to avoid carry.
6. Issue mtspr RTCU,Ry. Set upper register to time in seconds.
7. Issue mtspr RTCL,Rz. Set lower register to correct fraction of a second.

At the completion of this sequence, the RTC contains the correct time unless a delay such as an interrupt occurs during this sequence.

Reading the RTC

The content of either half of the RTC can be copied to a GPR using the mfspr instruction. This instruction does not change the content of the RTC. This is not a privileged access.

When the current time is required in a form that includes more than the upper or lower word of the RTC, the following procedure should be used:

1. Execute the following instruction sequence:

```
mfspr   Rx,RTCU
mfspr   Ry,RTCL
mfspr   Rz,RTCU.
```

2. IF Rz = Rx

   THEN the correct value has been obtained.

   ELSE go to step 1.

This procedure guarantees that the correct value is obtained.

Note: If the following instruction sequence is executed:

```
mtspr    RTCU,Rx
mfspr    Ry,RTCU,
```

then the contents of register Rx differ from the contents of register Ry by at most one unless the sequence is interrupted.

Decrementer

The *Decrementer* (DEC) is a decrementing counter that provides a mechanism for causing an external interrupt after a programmable delay. The period of bit 0 of DEC is approximately 4.3 seconds. See Figure 33.

Register	Description
DEC	Represents delay in nanoseconds

Figure 33. Decrementer

Note: All bits in DEC need not be implemented.

The driving frequency, insertion bit, and controls for the Decrementer must be implemented such that the following requirements are satisfied:

- The implemented bits function as a binary down counter.
- The operation of the RTC and the DEC are coherent; for example, both counters are driven by the same fundamental time base.
- The resolution of the initial implementation is 256 nanoseconds.
- If all bits are implemented, DEC(31) changes state each nanosecond.
- Loading a GPR from DEC has no effect on the counter. After the load, bits in the GPR corresponding to the unimplemented bits in the counter are 0's.
- Moving the content of a GPR to the DEC replaces the implemented bits of DEC with the associated bits of the GPR. Bits in the GPR corresponding to the unimplemented bits in the DEC are ignored.
- The Decrementer never stops running. When its contents are 0 and it is decremented, all the implemented bits are set to 1.
- Whenever bit 0 of DEC changes from 0 to 1, an interrupt request is signalled.
- If the DEC is altered by software and the content of bit 0 of DEC is changed from 0 to 1, an interrupt request is signaled.

Decrementer Interrupts

The *Decrementer Interrupt* is an External Interrupt and conforms to the specification as defined in "External Interrupt Mechanism" on page 1-67. The Decrementer Interrupt mechanism contains an EISBID located in the I/O space.

DEC EISBID Address

Segment Register Fields

BUID X'00'

Adapter Specific X'00000'

Effective Address

EA_{0-3} Must select a Segment register with a content as specified previously

E_{A4-31} X'0000020'

Decrementer Usage

The content of the Decrementer can be read or altered using the Special Purpose registers control instructions. The mtspr instruction is privileged when the DEC is the target register.

Setting the DEC

The content of the Decrementer is altered by software using the following mtspr instruction:

```
mtspr    DEC,Rx.
```

Note: If the execution of this instruction causes bit 0 of DEC to be changed from a value of 0 to a value of 1, an interrupt request is signalled.

Reading the DEC

The content of the Decrementer can be copied to a GPR by executing a mfspr instruction. Copying the Decrementer to a GPR has no effect on the Decrementer content or interrupt mechanism.

```
mfspr    Rx,DEC
```

Floating-Point Round to Single Model

The following describes the model for Floating-Point Round to Single-Precision instruction.

Floating-Point Round to Single Model

```
If FRB(1–11)<897 and FRB(1–63)>0 then
    Do
        If FPSCR(UE) = 0 then goto Disabled Exponent Underflow
        If FPSCR(UE) = 1 then goto Enabled Exponent Underflow
    End

If FRB(1–11)>1150 and FRB(1–11)<2047 then
    Do
        If FPSCR(OE) = 0 then goto Disabled Exponent Overflow
        If FPSCR(OE) = 1 then goto Enabled Exponent Overflow
    End

If FRB(1–11)>896 and FRB(1–11)<1151 then goto Normal Operand

If FRB(1–63) = 0 then goto Zero Operand

If FRB(1–11) = 2047 then
    Do
        If FRB(12–63) = 0 then goto Infinity Operand
        If FRB(12) = 1 then goto QNaN Operand
        If FRB(12) = 0 and FRB(13–63)>0 then goto SNaN Operand
    End
```

Disabled Exponent Underflow

```
sign ← FRB(0)
If FRB(1–11) = 0 then
    Do
        exp ← –1022
        frac ← b'0' || FRB(12–63)
    End
If FRB(1–11)>0 then
    Do
        exp ← FRB(1–11) – 1023
        frac ← b'1' || FRB(12–63)
    End
Denormalize operand:
    G || R || X ← b'000'
    Do while exp<–126
        exp ← exp + 1
        frac || G || R || X ← b'0' || frac || G || R or X
    End
FPSCR(UX) ← frac(24–52)||G||R||X>0
If frac(24–52)||G||R||X>0 then FPSCR(XX) ← b'1'
Round single(sign,exp,frac,G,R,X)
If frac = 0 then
    Do
        FRT(00) ← sign
        FRT(01–63) ← 0
        If sign = 0 then FPSCR(FPRF) ← "+zero"
```

```
            If sign = 1 then FPSCR(FPRF) ← "–zero"
        End

    If frac>0 then
        Do
            If frac(0) = 1 then
                Do
                    If sign = 0 then FPSCR(FPRF) ← "+normal number"
                    If sign = 1 then FPSCR(FPRF) ← "–normal number"
                End
            If frac(0) = 0 then
                Do
                    If sign = 0 then FPSCR(FPRF) ← "+denormalized number"
                    If sign = 1 then FPSCR(FPRF) ← "–denormalized number"
                End
            Normalize operand:
                Do while frac(0) = 0
                    exp ← exp–1
                    frac || G || R ← frac(1–52) || G || R || b'0'
                End
            FRT(0) ← sign
            FRT(1–11) ← exp + 1023
            FRT(12–63) ← frac(1–23) || 29*b'0'
        End
    Done
```

Enabled Exponent Underflow

```
FPSCR(UX) ← b'1'
sign ← FRB(0)
If FRB(1–11) = 0 then
    Do
        exp ← –1022
        frac ← b'0' || FRB(12–63)
    End
If FRB(1–11)>0 then
    Do
        exp ← FRB(1–11) – 1023
        frac ← b'1' || FRB(12–63)
     End
Normalize operand:
    Do while frac(0) = 0
        exp ← exp – 1
        frac ← frac(1–52) || b'0'
    End
If frac(24–52)>0 then FPSCR(XX) ← b'1'
Round single(sign,exp,frac,0,0,0)
exp ← exp + 192
FRT(0) ← sign
FRT(1–11) ← exp + 1023
FRT(12–63) ← frac(1–23) || 29*b'0'
If sign = 0 then FPSCR(FPRF) ← "+normal number"
If sign = 1 then FPSCR(FPRF) ← "–normal number"
Done
```

Disabled Exponent Overflow

```
FPSCR(OX) ← b'1'
FPSCR(XX) ← b'1'
If FPSCR(RN) = b'00' then (Round to Nearest)
    Do
        If FRB(0) = b'0' then
            Do
                FRT(0–63) ← x'7FF0000000000000'
                FPSCR(FPRF) ← "+infinity"
            End
        If FRB(0) = b'1' then
            Do
                FRT(0–63) ← x'FFF0000000000000'
                FPSCR(FPRF) ← "–infinity"
            End
    End
If FPSCR(RN) = b'01' then (Round Truncate)
    Do
        If FRB(0) = b'0' then
            Do
                FRT(0–63) ← x'47EF FFFF E000 0000'
                FPSCR(FPRF) ← "+normal number"
            End
        If FRB(0) = b'1' then
            Do
                FRT(0–63) ← x'C7EF FFFF E000 0000'
                FPSCR(FPRF) ← "–normal number"
            End
    End
If FPSCR(RN) = b'10' then (Round to +Infinity)
    Do
        If FRB(0) = b'0' then
            Do
                FRT(0–63) ← x'7FF0 0000 0000 0000'
                FPSCR(FPRF) ← "+infinity"
            End
        If FRB(0) = b'1' then
            Do
                FRT(0–63) ← x'C7EF FFFF E000 0000'
                FPSCR(FPRF) ← "–normal number"
            End
    End
If FPSCR(RN) = b'11' then (Round to –Infinity)
    Do
        If FRB(0) = b'0' then
            Do
                FRT(0–63) ← x'47EF FFFF E000 0000'
                FPSCR(FPRF) ← "+normal number"
            End
        If FRB(0) = b'1' then
            Do
                FRT(0–63) ← x'FFF0 0000 0000 0000'
                FPSCR(FPRF) ← "–infinity"
            End
    End
Done
```

Enabled Exponent Overflow

```
    sign ← FRB(0)
    exp ← FRB(1–11) – 1023
    frac ← b'1' || FRB(12–63)
    If frac(24–52)>0 then FPSCR(XX) ← b'1'
    Round single(sign,exp,frac,0,0,0)
Enabled Overflow:
    FPSCR(OX) ← b'1'
    exp ← exp – 192
    FRT(0) ← sign
    FRT(1–11) ← exp + 1023
    FRT(12–63) ← frac(1–23) || 29*b'0'
    If sign = 0 then FPSCR(FPRF) ← "+normal number"
    If sign = 1 then FPSCR(FPRF) ← "–normal number"
    Done

Zero Operand

    FRT(0–63) ← FRB(0–63)
    If FRB(0) = b'0' then FPSCR(FPRF) ← "+zero"
    If FRB(0) = b'1' then FPSCR(FPRF) ← "–zero"
    Done
```

Infinity Operand

```
FRT(0–63) ← FRB(0–63)
If FRB(0) = b'1' then FPSCR(FPRF) ← "–infinity"
Done
```

QNaN Operand

```
FRT(0–63) ← FRB(0–34) || 29*b'0'
FPSCR(FPRF) ← "QNaN"
Done
```

SNaN Operand

```
FPSCR(VXSNAN) ← b'1'
If FPSCR(VE) = 0 then
    Do
        FRT(0–11) ← FRB(0–11)
        FRT(12) ← b'1'
        FRT(13–63) ← FRB(13–34) || 29*b'0'
        FPSCR(FPRF) ← "QNaN"
    End
Done
```

Normal Operand

```
sign ← FRB(0)
exp ← FRB(1–11) – 1023
frac ← b'1' || FRB(12–63)
If frac(24–52)>0 then FPSCR(XX) ← b'1'
Round single(sign,exp,frac,0,0,0)
If exp>+127 and FPSCR(OE ) = 0 then go to Disabled Exponent Overflow
If exp>+127 and FPSCR(OE) = 1 then go to Enabled Overflow
FRT(0) ← sign
FRT(1–11) ← exp + 1023
FRT(12–63) ← frac(1–23) || 29*b'0'
If sign = 0 then FPSCR(FPRF) ← "+normal number"
If sign = 1 then FPSCR(FPRF) ← "–normal number"
Done
```

Round Single (sign, exp, frac, G, R, X)

```
inc ← b'0'
lsb ← frac(23)
gbit ← frac(24)
rbit ← frac(25)
xbit ← frac(26–52)||G||R||X>0
If FPSCR(RN) = b'00' then
    Do
        If sign || lsb || gbit || rbit || xbit = b'x11xx' then inc ← b'1'
        If sign || lsb || gbit || rbit || xbit = b'x011x' then inc ← b'1'
        If sign || lsb || gbit || rbit || xbit = b'x01x1' then inc ← b'1'
    End
If FPSCR(RN)=b'10' then
    Do
        If sign || lsb || gbit || rbit || xbit = b'0x1xx' then inc ← b'1'
        If sign || lsb || gbit || rbit || xbit = b'0xx1x' then inc ← b'1'
        If sign || lsb || gbit || rbit || xbit = b'0xxx1' then inc ← b'1'
    End
If FPSCR(RN)=b'11' then
    Do
        If sign || lsb || gbit || rbit || xbit = b'1x1xx' then inc ← b'1'
        If sign || lsb || gbit || rbit || xbit = b'1xx1x' then inc ← b'1'
        If sign || lsb || gbit || rbit || xbit = b'1xxx1' then inc ← b'1'
    End
frac(0–23) ← frac(0–23) + inc
If carry out=1 then
    Do
        frac(0–23) ← b'1' || frac(0–22)
        exp ← exp + 1
    End
FPSCR(FR) ← inc
FPSCR(FI) ← gbit or rbit or xbit
Return
```

Floating-Point Integer Convert Model

This section describes the conversion of the floating-point double precision value contained in register FRB into an integer or a special number if the conversion cannot be successfully completed. This function converts a 64-bit floating-point value to a 32-bit integer. Whether the conversion is successful or is an exception case, the high order 32 bits of RT are undfined. If the conversion is successful, the low order 32 bits of RT contain the integer resulting from the conversion. The 32 undefined bits are indicated by "xxxx xxxx" in hex representations of the 64-bit register.

Floating-Point Integer Conversion

The follow segments describe the expected result based on the content of FRB:

```
If [Floating Convert to Integer and Round]
    Then round_mode ← FPSCR(RN)

If [Floating Convert to Integer and Round toward Zero]
    Then round_mode ← b'1'

FPSCR(FPRF) ← "undefined"

If FRB(1:11) = 2047 and FRB(12:63) = 0 then goto Infinity Operand
If FRB(1:11) = 2047 and FRB(12) = 0 then goto SNaN Operand
If FRB(1:11) = 2047 and FRB(12) = 1 then goto QNaN Operand
If FRB(1:11) > 1087 then goto Large Operand

sign ← FRB (0)
If FRB(1:11) > 0 then exp ← FRB(1:11) – 1023    /** exp – bias **/
If FRB(1:11) = 0 then exp ← –1022
If FRB(1:11) > 0 then frac(0:63) ← b'01' || FRB(12:63) || b'00000000000'    /** normal **/
If FRB(1:11) = 0 then frac(0:63) ← b'00' || FRB(12:63) || b'00000000000'    /** denormal **/

gbit || rbit || xbit ← b'000'

Do i = 1, 64–exp
    frac(0:63) || gbit || rbit || xbit ← b'0' || frac (0:63) || gbit || rbit or xbit
End

If gbit or rbit or xbit then FPSCR(XX) ← 1
```

Round Integer (sign, frac, gbit, rbit, xbit, round_mode)

```
Round Integer (sign, frac, gbit, rbit, xbit, round_mode)
If sign = 1 then frac(0:63) ← –frac (0:63) + 1
    If frac (0:63) > +2**(31–1) then goto Large Operand
    If frac (0:63) < –2**(31)   then goto Large Operand

FRT ← x'xxxx xxxx' || frac (32:63)    /** where x'xxxx xxxx' is undefined **/

Done (exit conversion)
```

```
Round Integer (sign, frac, gbit, rbit, xbit, round_mode)
{
    inc ← b'0'
    If round_mode = b'00' then
        Do
            If sign || frac(63) || gbit || rbit || xbit = b'x11xx' then inc ← 1
            If sign || frac(63) || gbit || rbit || xbit = b'x011x' then inc ← 1
            If sign || frac(63) || gbit || rbit || xbit = b'x01x1' then inc ← 1
        End
    If round_mode = b'10' then
        Do
            If sign || frac(63) || gbit || rbit || xbit = b'0x1xx' then inc ← 1
            If sign || frac(63) || gbit || rbit || xbit = b'0xx1x' then inc ← 1
            If sign || frac(63) || gbit || rbit || xbit = b'0xxx1' then inc ← 1
        End
    If round_mode = b'11' then
        Do
            If sign || frac(63) || gbit || rbit || xbit = b'1x1xx' then inc ← 1
            If sign || frac(63) || gbit || rbit || xbit = b'1xx1x' then inc ← 1
            If sign || frac(63) || gbit || rbit || xbit = b'1xxx1' then inc ← 1
        End
    frac(0:63) ← frac(0:63) + inc
    FPSCR(FR) ← inc
    FPSCR(FI) ← gbit or rbit or xbit
    Return    /* end of Round Integer */
}
```

Infinity Operand

```
Infinity
{
If the content of FRB is a representation of infinity, the following is required:
    1.  FPSCR(FR, FI, VXCVI) ← b'001'

    2.  If FPSCR(VE) = 0
            THEN DO
                If the sign = 0
                    then do
                        FPSCR(FPRF) ← "+infinity"
                        FRT ← x'xxxx xxxx 7FFF FFFF'
                    end do

                If the sign = 1
                    then do
                        FPSCR(FPRF) ← "–infinity"
                        FRT ← x'xxxx xxxx 8000 0000'
                    end do
            END DO
Done (exit conversion)
}
```

SNaN Operand

```
SNaN Operand
{
    If the content of FRB is an SNaN, the following is required:
        FPSCR(FR, FI, VXCVI) ← b'001'
        If FPSCR(VE) = 0
            THEN DO
                FPSCR(FPRF) &ldarrow. "quiet NaN"
                FRT &ldarrow. ← x'xxxx xxxx 8000 0000'
            END DO
    Done (exit conversion)
}
```

QNaN Operand

```
QNaN Operand
{
    If the content of FRB is a QNaN, the following is required:
        FPSCR(FR, FI, VXCVI) ← b'001'
        If FPSCR(VE) = 0
            THEN DO
                FPSCR(FPRF) &ldarrow. "quiet NaN"
                FRT &ldarrow. ← x'xxxx xxxx 8000 0000'
            END DO
    Done (exit conversion)
}
```

Large Operand

```
Large Operand
{
    If the content of FRB, rounded as indicated by the instruction being executed, is too
    large to be represented in 32 bits, the following is required:
        FPSCR(FR, FI, VXCVI) ← b'001'
        If FPSCR(VE) = 0
            then if the sign = 0
                then FRT &ldarrow. ← x'xxxx xxxx 7FFF FFFF'
                else FRT &ldarrow. ← x'xxxx xxxx 8000 0000'
    Done (exit conversion)
}
```

I/O Space Rules

The following rules should be adhered to when addressing I/O segments using loads and stores:

- All references, both loads and stores, must be generated.
- The order of the references to shared variables must not be change by the compiler. This is with respect to all shared variables, not just the same shared variable.
- No references can be moved outside of their basic block (for example, before an if test or outside of a loop).

- Multiple references to adjacent locations cannot be combined into a single reference (for example, a load byte from 1fe combined with a load byte from 1ff to create a load halfword from 1fe).
- Read-modify-write cannot be supported and should produce a compile time error. The programmer must be forced to explicitly program to the underlining storage classes (character, halfword, word) for all references.

Serializing Semantics of Various Instructions

Some Serialization Cases

In order to arrive at the definitions of ics and dcs instructions, the following cases where synchronization is required were considered:

- Synchronization on local I/O operations:

  Assume memory control registers are being updated. The sequence of instructions occur as follows:

  ```
  Store          ram bank control
  <sync>.
  ```

  In this case the synchronization can be taken care of by an ics instruction. The ics must wait for the store to complete (at this time it is removed from the PCS).

- Instruction modification:

  The following is a possible sequence:

  ```
  Store          (changed instruction)
  clf
  <sync> (wait for store-back to complete, invalidate prefetch buffers)
  Branch (to changed instruction).
  ```

  This synchronization is accomplished by issuing a dcs instruction first, followed by an ics instruction. The entire sequence then becomes the following:

  ```
  Store          (changed instruction)
  clf
  dcs
  ics
  Branch (to changed instruction).
  ```

  The dcs instruction waits for the store-back to main memory to finish at the Fixed-Point unit. The ics forces the Instruction Cache Unit to wait until the dcs is complete. Any prefetched instructions are invalidated, and the instruction following the ics is fetched again. The fetching of the branch target causes an instruction cache miss and the new version of the line is fetched from memory.

- Page in:

  ```
  cli
  <multiple clis>
  <sync>
  (Invalidate prefetch instructions and wait for last cli to complete).
  ```

This case is also handled by an ics instruction. The cli and clf instructions are placed on the Program Counter Stack (PCS) (see the following). The ics instruction waits for the PCS to empty before it can complete.

- Page out:

  clf
  <multiple clfs>
  <sync> (serialize Fixed-Point Unit, wait for store-back of last clf)
  <start I/O>.

  In this case the synchronization is handled by a dcs instruction. The Instruction Cache Unit is not synchronized; it continues dispatching instructions beyond the dcs instruction as there is no need to synchronize the Instruction Cache Unit.

Instruction Cache Synchronize and Data Cache Synchronize Definitions

The following sections describe the instruction cache synchronize (ics) and data cache synchronize (DCS) instructions.

ics Instruction

The ics instruction should have the following semantics:

- Any prefetched instructions are discarded.
- The PCS is emptied. The PCS maintains a hardware list of outstanding instructions in the Fixed-Point Unit which can cause an interrupt. They include loads, stores, and traps.
- Any outstanding operations from the following list must have executed (meaning that none of the following instructions causes an interrupt, and are completely executed with respect to the state of processor registers, but perhaps not memory):
  - dcs
  - tlbi
  - mtsr and mtsri
  - cli
  - dclst, dclz, and clf.

  (To the point they cannot interrupt by way of the PCS, the line may not be valid in main memory for clf and dclst, and the cache line may not be entirely zeroed for dclz.)

The previous three conditions are referred to as the three serializing operations. Upon encountering ics, the Instruction Cache Unit waits until these conditions are satisfied before considering any subsequent instructions for dispatch.

There are actually two problems related to SDR1 and SDR0. The primary problem is what the correct behavior of ics should be relative to these SPR moves. The second is what the Instruction Cache Unit should do with respect to misses for prefetched instructions when these registers are in the process of being updated.

There is a delay between the time an mtspr. SDR0/SDR1 is dispatched by the Instruction Cache Unit and the time it is actually executed by the Fixed-Point Unit. During this time the Instruction Cache Unit must be prevented from presenting a translation request to the Fixed-Point Unit, otherwise an incorrect translation could possibly be performed. We recommend that this case be handled in the following way.

Update SDR0 and SDR1 in real mode only. In this case there is no possibility of an ITLB miss, so an incorrect translation cannot be performed.

If this is unacceptable and software wishes to update SDR0 and SDR1 in virtual mode, then an ics must follow the mtspr SDR0/SDR1 in order to ensure correct operation. (This solution will not work if the ics and mtspr are on different pages and ics is on a page affected by the new value of SDR0/SDR1.)

dcs Instruction

The dcs instruction waits for all outstanding data cache operations (clf, dclst, dclz) to complete. (By virtue of the present design of the Fixed-Point Unit, cli, tlbi, mtsr, and mtsri will all have completed prior to the dcs completing.) The dcs instruction does not synchronize the Instruction Cache Unit. (However, an interlock bit is set in order to allow ics to interlock until the dcs instruction completes.)

Other Instructions Possibly Requiring Serialization

The semantics of other serializing or potentially serializing instructions are listed as follows:

- svc

  As part of the execution of this instruction all three serializing operations listed previously for ics are performed. (Although the svc instruction is not presently defined as a serializing instruction, the initial implementation implements it as such.) In addition, the SVC cannot be executed until the Link register is not interlocked (if the LK bit is set) and the Count register and MSR are not interlocked.

- mtmsr

  The mtmsr instruction will not be dispatched until the MSR is not interlocked. When it is dispatched no subsequent instructions will be dispatched until the MSR has been updated. At this point any prefetched instructions will be invalidated, and the instruction following mtmsr will be refetched using the new MSR value. (Currently it is not necessary to wait for the PCS to empty because the Fixed-Point Unit will not be able to perform the mtmsr until all instructions that could cause an interrupt in the Fixed-Point Unit have been completed.)

- rfi

  All three serializing operations will be performed. Additionally, the rfi instruction will not execute until SRR0, SRR1, and the MSR are not interlocked.

- rfsvc

  All three serializing operations will be performed. The rfsvc will not be executed until the Link register, the Count register, and the MSR are not interlocked.

- tlbi

  This instruction does not serialize the Instruction Cache Unit. An explicit ics must be issued if TLB entries pertaining to the page from which instructions are being fetched or pre-fetched are being invalidated.

- mtsr and mtsri

  These instructions do not serialize the Instruction Cache Unit. Instructions past the mtsr or mtsri instructions are dispatched. Prefetched instructions are not invalidated. If a Segment register from which instructions are being fetched is to be updated, the Segment register update must be followed by an ics instruction.

  These instructions are self-serializing with respect to data references.

- Load/Store to I/O

  There is no known serialization since all of these operations must complete before any subsequent operations are executed by the Fixed-Point Unit, and we are not aware of any effect of these operations on the Instruction Cache Unit which requires implicit serialization.

- clf and cli

  These instructions do not serialize the Instruction Cache Unit. An ics instruction must be issued to cause the Instruction Cache Unit to wait until all outstanding clf/cli operations have been executed, and to fetch again any fetched instructions.

- dclz and dclst

  These instructions cause no serialization in the instruction cache unit, but are self-serializing in the Fixed-Point Unit.

- mtspr TID

  This instruction is presently self-serializing in the Fixed-Point Unit so it requires no special handling by software.

- mtspr SDR 0

- mtspr

  Instructions which move to SPRs in the Instruction Cache Unit, such as SRR0, SRR1, LR, and CTR, are all handled by a standard interlock scheme. When the instruction is dispatched, an interlock bit for the affected register is set. When the data returns from the Fixed-Point Unit, the interlock bit is reset. No subsequent read or write operation to a register can be performed while the interlock bit is set. These instructions are not serializing.

Chapter 2. System I/O Structure

Chapter Contents

Description

This chapter describes the Input/Output (I/O) architecture. General I/O bus support functions for Load and Store instructions, interrupt, and channel control are provided by the I/O Channel Controller (IOCC). A number of feature I/O slots are associated with the IOCC for pluggable I/O devices. Also attached to the I/O bus, but not occupying feature slots, is the Standard I/O. See "System I/O and Standard I/O" on page 2-84.

The IOCC design allows certain variations of function and performance to optimize its use across multiple machine environments. The specific personalization is established with the contents of the IOCC Configuration register. (See "IOCC Configuration Register" on page 2-74) and "Implementation Details" on page 2-86.)

Reasonable efforts were made to implement this architecture correctly and completely. However, the implementations may deviate to some extent from the I/O architecture, documented in this chapter. The specifics of the various implementation deviations are documented in the "Implementation Details" on page 2-86 or in the I/O architecture implementation details section in the product-specific manual.

Figure 34 shows the logical view of the IOCC in the units.

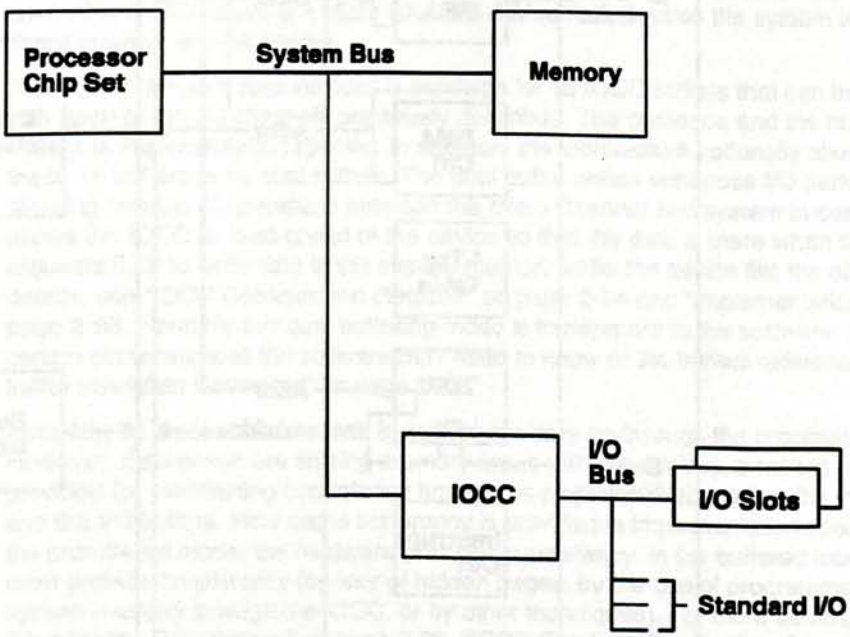
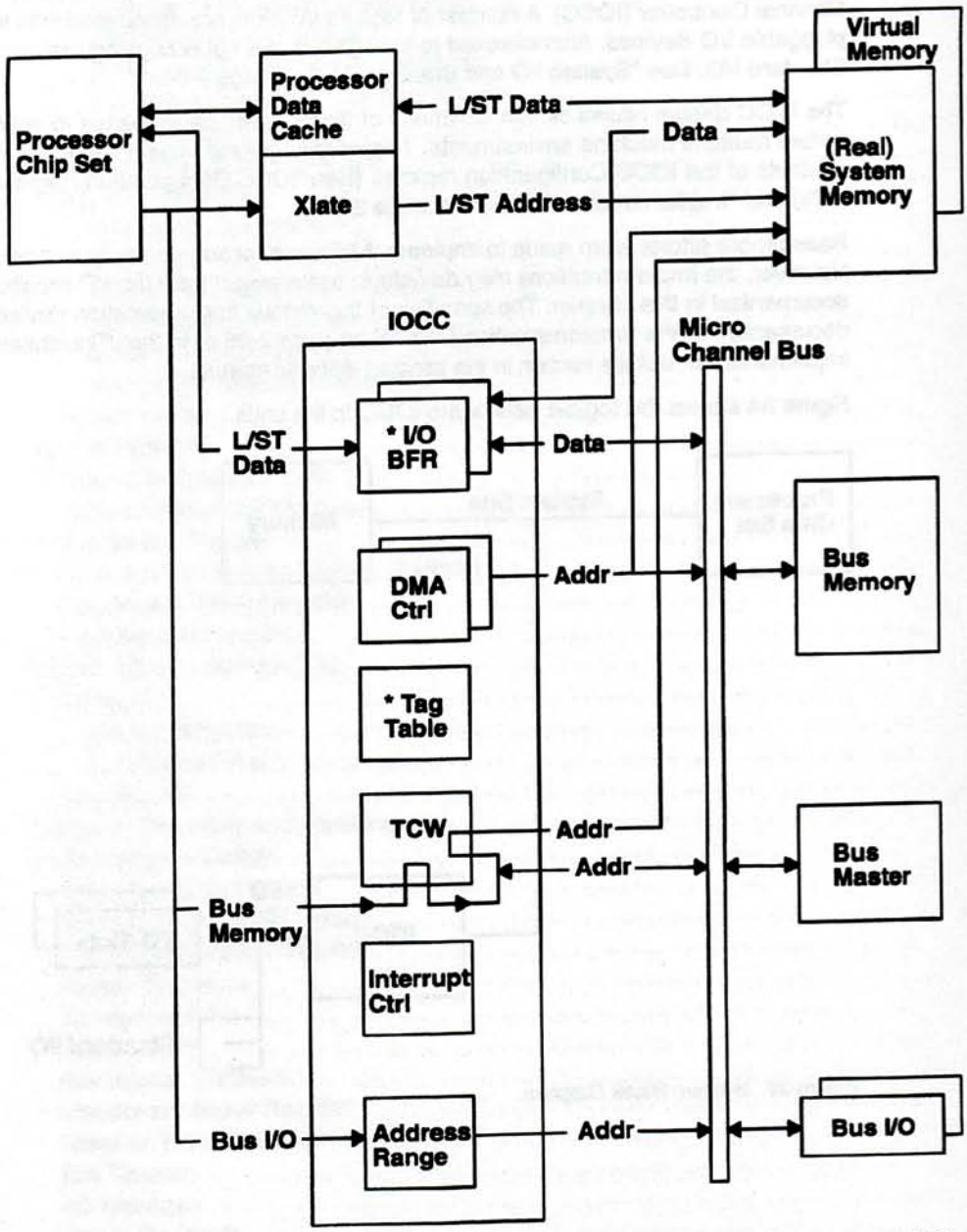


Figure 34. System Block Diagram

System Structure

Figure 35 shows a more detailed logical view of the IOCC. Functions provided by the IOCC include data buffering, address translation, access protection, direct memory access (DMA), and interrupt support.



Note: * May be implementation specific. (See "Implementation Details" on page 2-86).

Figure 35. Programming Model

The operating system can access all system facilities, for example, virtual memory, system memory, bus I/O, bus memory, and the IOCC. The IOCC contains special facilities needed by the system for translation, protection, and other functions.

Problem state programmers are normally restricted to virtual memory. The virtual address is always mapped to system memory by way of the translation mechanism associated with the processor chip set.

For certain applications, the operating system also grants conditional access authority to the bus I/O and bus memory. Accesses to bus memory and bus I/O devices are checked for proper access authority, restricting user programs to access only the devices that they are authorized to use. Accesses to bus I/O are verified by an address range check. Accesses to bus memory are verified by way of a key in the translate control word (TCW) table described in "Translation, Protection, and TCW Table" on page 2-33.

The I/O architecture includes the definition of 16 independent I/O channels. One channel (X'F') is used by the system master for Load and Store transfers, leaving 15 that can be programmed for bus master transfers. The number of channels that can be programmed for DMA slave transfers is implementation specific. (See "IOCC Configuration Register" on page 2-74 and "Implementation Details"on page 2-86.)

A bus master is a Micro Channel device that contains its own direct memory access controller. A DMA slave is a Micro Channel device that requires the system to provide the direct memory access control.

The I/O architecture also includes a provision for 16 IOCC buffers that can be associated with each of the I/O channels previously described. The presence and the number of IOCC buffers is implementation specific. In addition, the architecture optionally allows for each of these 16 buffers to be dual buffers. The dual buffer option enhances I/O performance by allowing overlap of operations between the Micro Channel and system buses. The option allows the IOCC to read-ahead of the device so that the data is there when the device requests it, or to write data to the system memory while the device fills the other buffer. For details, see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86. Normally this dual buffering mode is transparent to the software. However, under certain circumstances the software may need to know of the buffers existence. See "next buffer invalidate Command" on page 2-68.

Normally, all processor accesses to system memory go through the processor data cache. However, if accesses are sharing memory areas with I/O devices, a means must be provided for maintaining consistency among the processor data cache, the system memory, and the I/O buffers. How cache coherency is provided is implementation specific. Briefly, in the unbuffered mode, the hardware provides consistency. In the buffered mode, the software must provide consistency (by way of hidden pages, by the use of programmed I/O (PIO) to system memory through the IOCC, or by other techniques). For more details, see "Maintaining Consistency" on page 2-36, "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86. All caches can be visible to programmers, including selected application level programmers.

A bus master on the I/O bus accesses bus memory and bus I/O, and if mapped, system memory. Pages in the bus memory address space are mapped to system memory by way of the TCW table and by a bit in each Channel Status register indicating the target (bus or system memory) of the access. Mapped pages are checked for proper access authority before allowing an access to proceed. Since the IOCC cannot intercept or stop accesses from a bus master to bus attached memory or bus I/O devices, no access checking is performed when a bus master addresses devices on the I/O bus.

The DMA slave controller provides a convenient mechanism for moving data between an I/O device and system or bus memory. It provides addressing and control functions on behalf of the I/O device. Two methods for providing addresses for the DMA slave operations are supported in the architecture. In the first, memory addresses are obtained from a tag table in the IOCC. This table provides translation facilities similar to the System/370 indirect address word list, with additional capabilities allowing data chaining down to the byte level. In the second method, a TCW table provides the Real Page Number (RPN) used along with an offset as the memory address. Both methods are described in more detail later in this document. For implementation specific details, see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86.

Virtual Memory

Virtual memory is a large address space containing logical system objects such as programs and data. Each object is assigned a unique address in the virtual memory space at the time of creation. This address is used thereafter to reference that object.

Virtual memory objects are mapped to system memory on a demand basis. At the time of reference by a system or user program, the translate unit associated with the processor chip set verifies whether that object is currently in system memory. If so, the unit supplies the appropriate (real) memory address. If the object is not in system memory, the operating system is called to obtain the requested object, place it in system memory, and update the tables used by the translate unit. The original faulting instruction is then retried and control is returned to the original system or user program. As long as the (virtual) access does not have any real-time dependencies, this demand mapping is transparent.

System Memory

System memory is closely associated with the processor chip set complex. The system architecture provides for up to 4G bytes of system memory.

Bus master and DMA slave operations to this memory neither synchronize nor update the processor data cache or Page Frame Table (PFT). Without proper programming precautions, this can cause the processor data cache and its associated system memory to be inconsistent, resulting in the loss or corruption of data (for example, when the processor chip set and an I/O device both attempt to access the same memory area). For more details, see "Maintaining Consistency" on page 2-36.

Bus Memory

I/O bus memory is the memory that logically resides on the I/O bus. The I/O bus includes 32 address bits, providing up to 4G bytes of addressability. PC family I/O buses utilize disjointed address spaces for bus memory and I/O devices. In the system units, these two address spaces are mapped together as shown in Figure 46 on page 2-21. This address space is differentiated from the I/O address space by an address decode. I/O bus memory is referenced when the address is above 64K bytes. Processor accesses to this memory space do not go through the processor data cache and do not suffer from the cache consistency problems described in the preceding section, "System Memory."

Bus memory is generally packaged on feature I/O cards and is associated with specific devices. Devices are generally mapped into the bus memory space when they have large addressability requirements, such as video display buffers and floating-point work space. Any bus master on the I/O bus has unconditional access to other devices on the Micro Channel I/O bus. As such, access to bus memory is unprotected.

Bus memory references are redirected to system memory by way of the TCW mechanism, the Channel Status register mapping bit, and, for systems that implement the optional Bus Mapping registers, by way of the Bus Mapping registers. Refer to the "Translation, Protection, and TCW Table" on page 2-33 for a description of this mapping process. Accesses to system memory are translated and checked for appropriate authority before allowing them to proceed. If allowed to proceed, this mapping of bus addresses to system memory is transparent to the requesting bus master or DMA slave. Special rules must be followed to guarantee the consistency of this memory if it is shared with the processor chip set. See "Maintaining Consistency" on page 2-36 for a description of these rules.

Bus I/O

The I/O bus includes a special address space for accessing I/O Control registers. This address space is mapped together with the bus memory and is referenced when the address is within the lower 64K bytes. It includes16 address bits and provides up to 64K bytes of addressability. I/O devices do not decode address bits A31 to A16 and these address bits are considered undefined relative to I/O devices. Note that the addressing nomenclature on the I/O bus follows the Micro Channel format shown in Figure 36 on page 2-8.

IOCC Control Registers

IOCC Control registers are special facilities managed by the system supervisor that control all aspects of the Load and Store instructions, channel, and interrupt operations. They are only accessible to Load and Store instructions from the system processor. They are addressed in a disjoint address space inaccessible to I/O bus devices. This address space is defined so that it can be mapped onto the I/O bus, providing flexible implementation in distributing IOCC control facilities across multiple chip packages. Refer to the "Special Facilities" on page 2-72 for a description of these registers.

Data Security

The system unit is intended to be used in shared environments and contains mechanisms to maintain data security. The IOCC supports attachment of user-supplied I/O devices and device drivers. The IOCC includes extensive hardware and operating system mechanisms to insulate the system and other users from them. All accesses to memory or the I/O bus are checked to verify that the user has authority to use that resource. Shared resources, such as IOCC or memory buffers, are controlled (for example, zeroed) so that no task gets access to some other task's data.

Bit and Byte Numbering Conventions

This section describes the processor and Micro Channel bus notations used for addressing bits, bytes, and multibyte fields, as well as the effects of these notations on the IOCC architecture.

Processor and Bus Notation

Two different methods are used to address the individual bytes in a multibyte scalar (numeric value) field. The methods differ in whether the field is addressed from the most-significant byte (the "big" end) or the least-significant byte (the "little" end).

The *big-endian* notation addresses scalar fields in ascending order from left to right. This results in the most-significant byte (MSB) always having the lowest address. This practice provides consistency in addressing that is independent of the word size of the machine. Bits are always numbered from left to right. This notation is used in all processor, channel, and serial protocol descriptions.

The *little-endian* notation reverses both bit and byte addressing for scalar fields. This notation is used in the Micro Channel architecture.

Regardless of which method is used to address scalar fields, all systems address string fields the same way, with the MSB having the lowest address.

Figure 36 shows the differences between big-endian and little-endian notation.

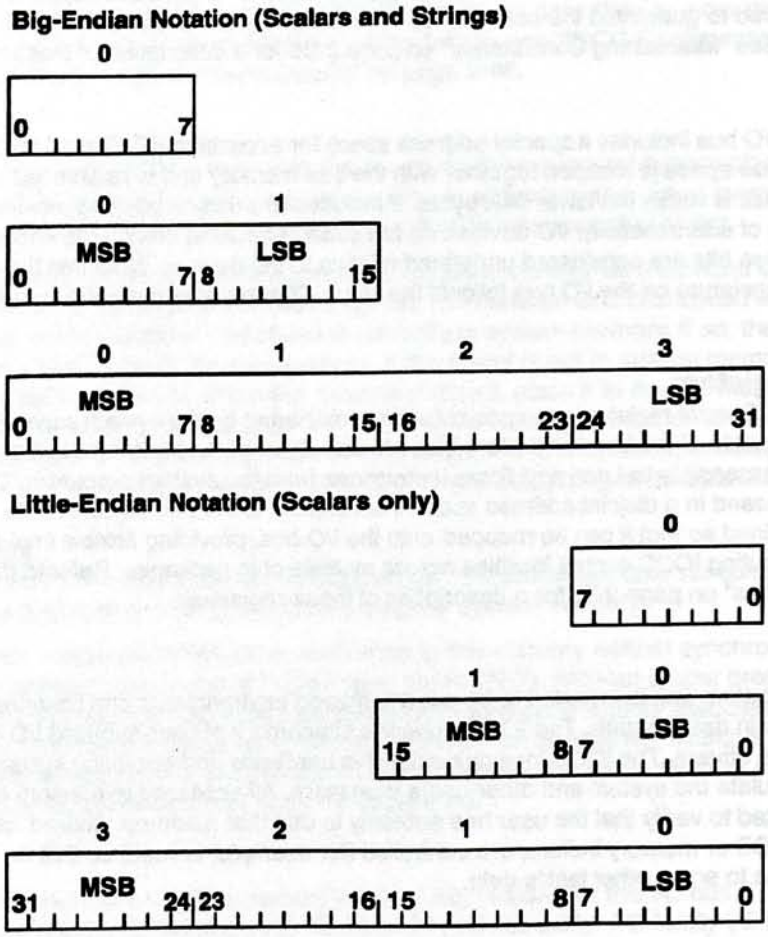
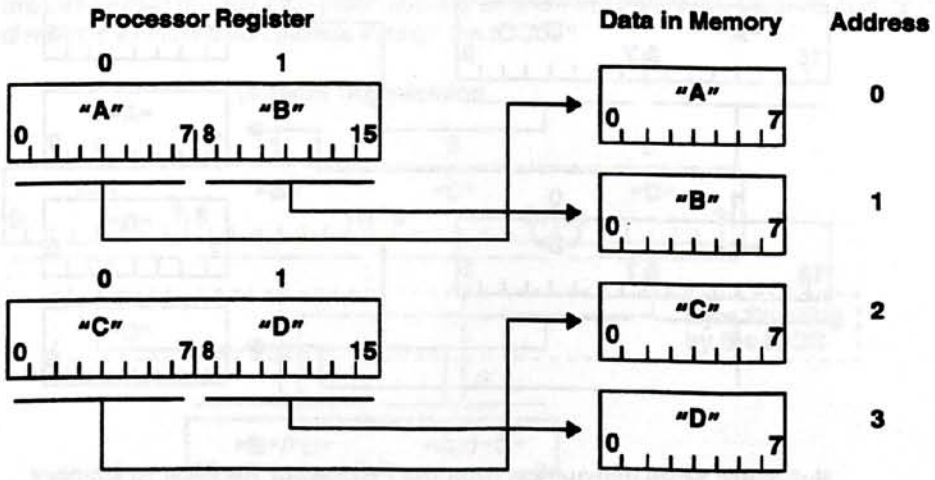


Figure 36. Data Addressing and Bit Numbering Notations

The little-endian practice of numbering bytes in ascending order from right to left results in the most significant byte of a word having the highest address. This poses problems in byte ordering on 2- or 4-byte buses. For byte strings such as text to be compatible across different word lengths and between different systems, the strings must be organized with the most significant byte having the lowest address. Figure 37 on page 2-9 shows the address consistency with the big-endian notation. Figure 38 on page 2-10 shows the address inconsistency when using the little-endian notation. With the little-endian numbering scheme,

there is no consistency in addressing across the various word sizes; two half-word stores produce a different result in memory than one full-word store.

Two Half-Word Store Instructions from the Processor Register to Memory



Full-Word Store Instruction from the Processor Register to Memory

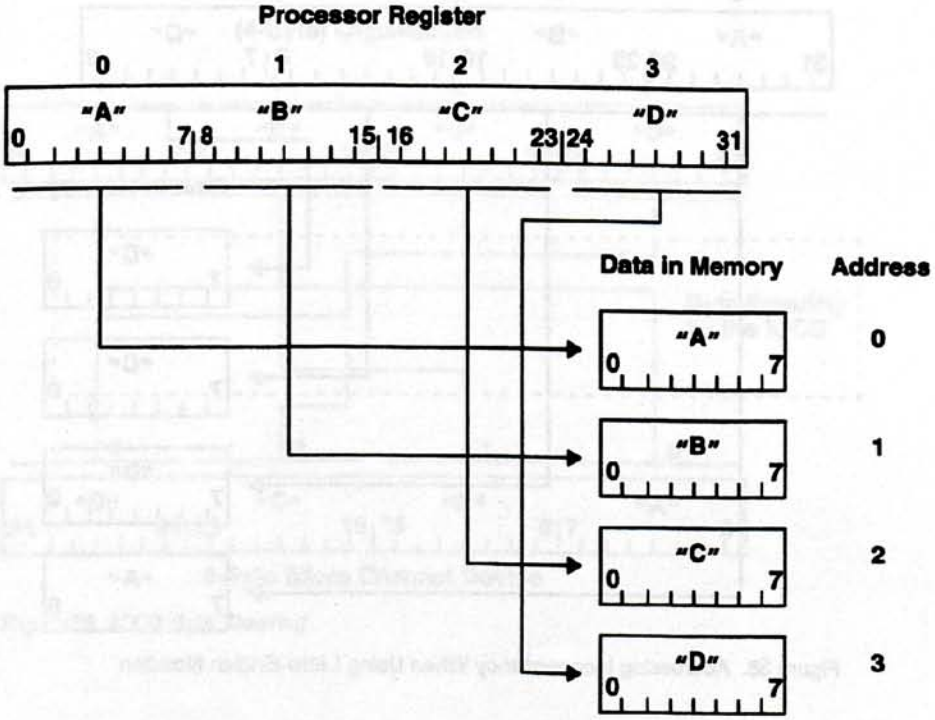


Figure 37. Addressing Consistency Using Big-Endian Notation

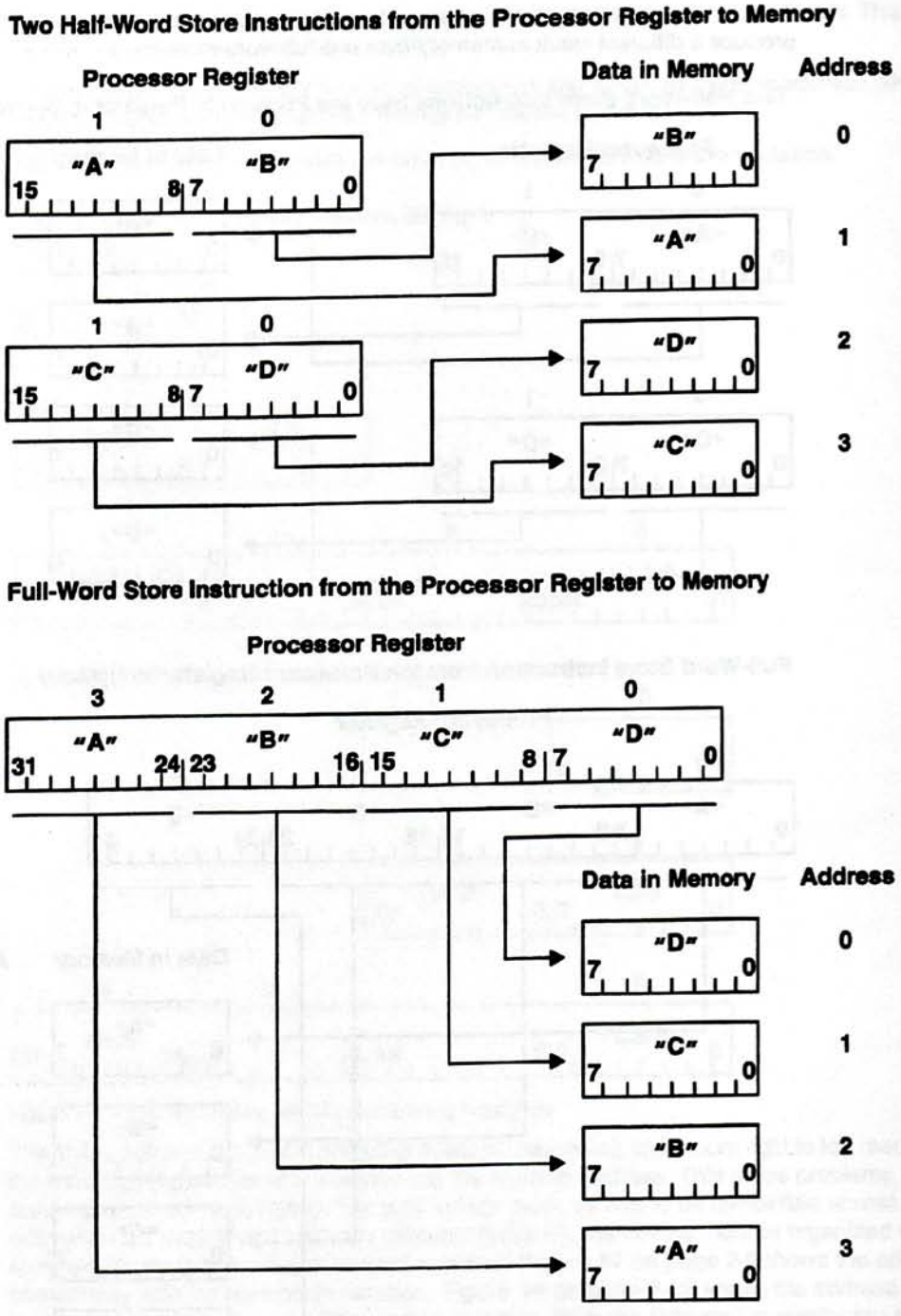


Figure 38. Addressing Inconsistency When Using Little-Endian Notation

IOCC Byte Steering

The compilers use big-endian addressing notation to handle data in the system unit. To match the little-endian notation of the Micro Channel bus, the bytes from the system must be steered to the appropriate bytes on the Micro Channel bus. The IOCC and the system board are designed to provide byte-order steering as shown in Figure 39. Steering occurs in both directions as information passes through the IOCC.

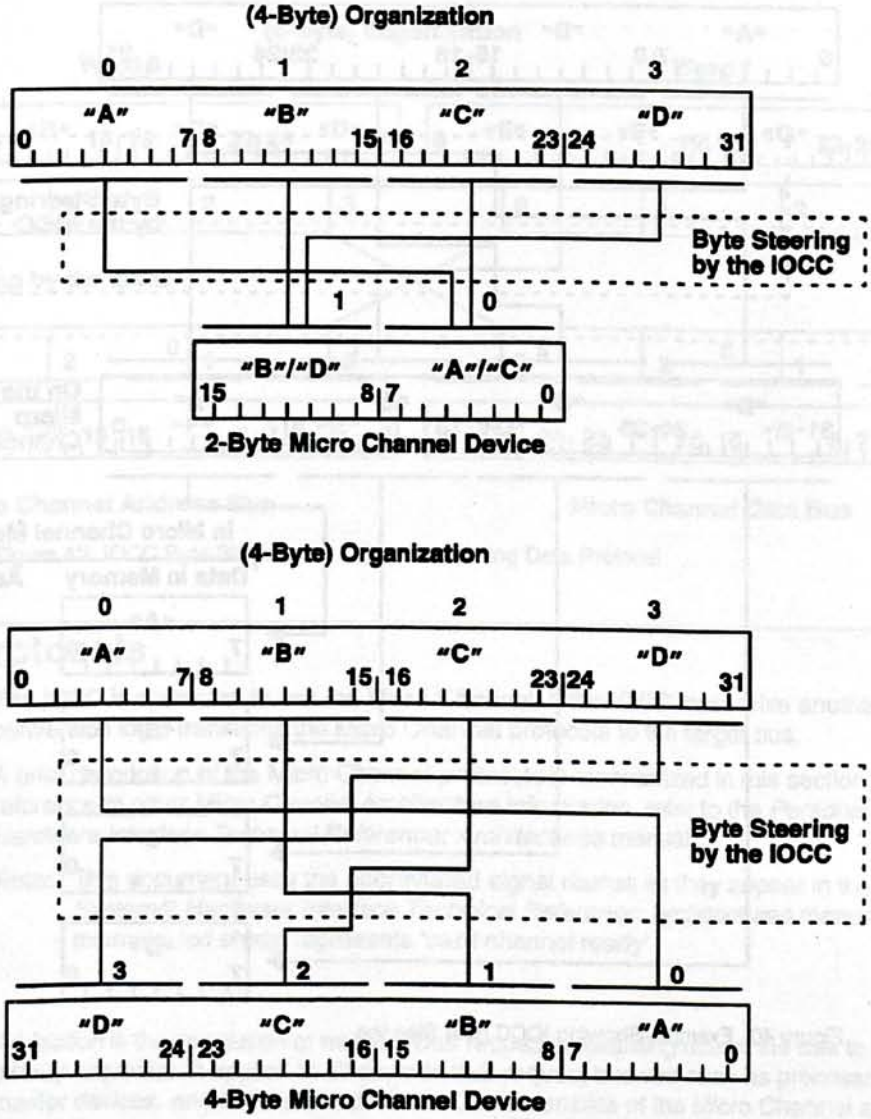


Figure 39. IOCC Byte Steering

The I/O data bits require renaming but otherwise maintain a one-to-one ordering with standards.

Combining both examples gives the byte steering shown in Figure 40.

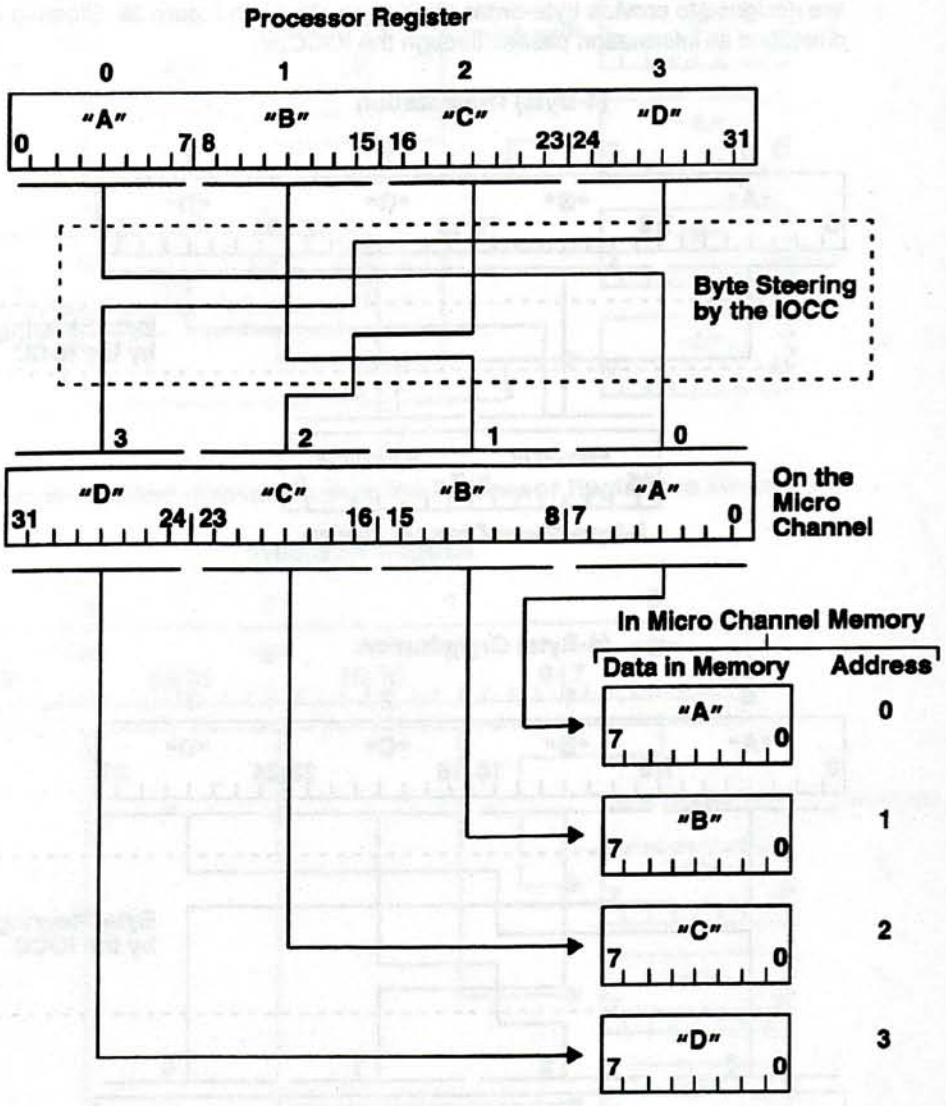


Figure 40. Example Showing IOCC Byte Steering

This architecture optionally allows for the implementation of the 8-byte Micro Channel Streaming Data protocol. For implementations that support the optional 8-byte Micro Channel Streaming Data operations (see "Implementation Details" on page 2-86 for details on which systems support the 8-byte Streaming Data protocol), the words are steered appropriately, as well as the bytes within the words, as shown in Figure 41. Steering occurs in both directions as information passes though the IOCC.

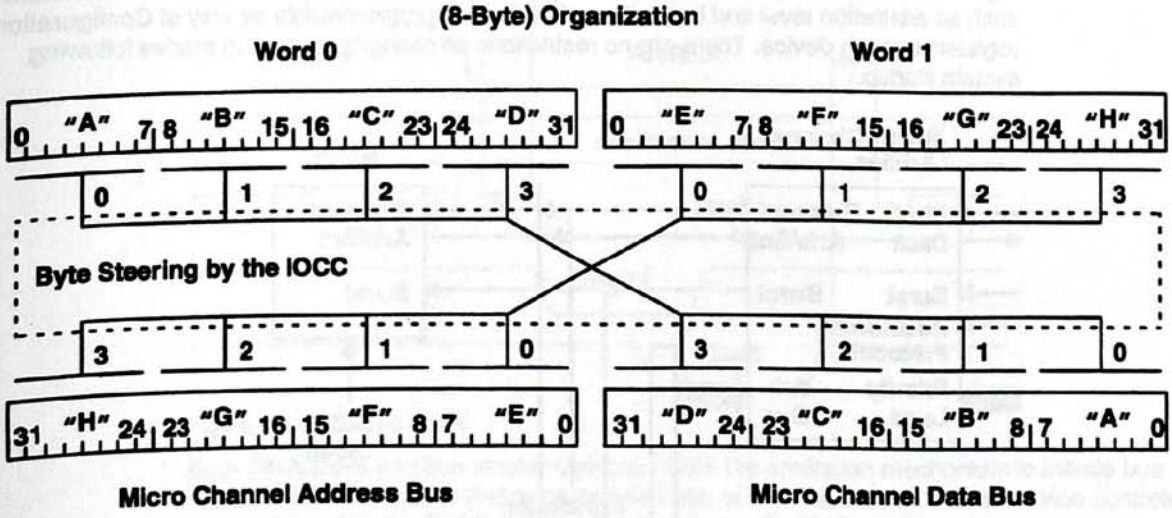


Figure 41. IOCC Byte Steering for 8-Byte Streaming Data Protocol

I/O Bus Protocols

The IOCC is optimized to use the Micro Channel. If the IOCC must drive another bus, conversion logic translates the Micro Channel protocols to the target bus.

A brief description of the Micro Channel protocols is summarized in this section. For reference to other Micro Channel architecture information, refer to the *Personal System/2 Hardware Interface Technical Reference: Architectures* manual.

Note: This document uses the abbreviated signal names as they appear in the *Personal System/2 Hardware Interface Technical Reference: Architectures* manual. For example, 'cd chrdy' represents 'card channel ready'.

Arbitration

Arbitration is the resolution of multiple bus requests, awarding use of the bus to the highest priority requester. It applies to all devices that request bus use such as processors, bus master devices, and DMA slave devices. Characteristics of the Micro Channel arbitration mechanism include:

- One to 16 bus masters
- Multidrop (dot-OR) mechanism
- Parallel prioritization
- Asynchronous operation
- Cycle-by-cycle arbitration
- Programmable priority levels
- Programmable fairness mode

- Mixable linear and fairness modes
- Preemptive burst capability
- Multiple bus extension.

The arbitration mechanism distributes prioritization among the arbiters but retains control and clocking functions within the IOCC. Bus arbitration timing is programmable and is established by a field in the IOCC Configuration register.

Figure 42 shows the typical device arbiters and their relationship in the system. Parameters such as arbitration level and burst characteristics are programmable by way of Configuration registers in each device. There are no restrictions on changing operating modes following system startup.

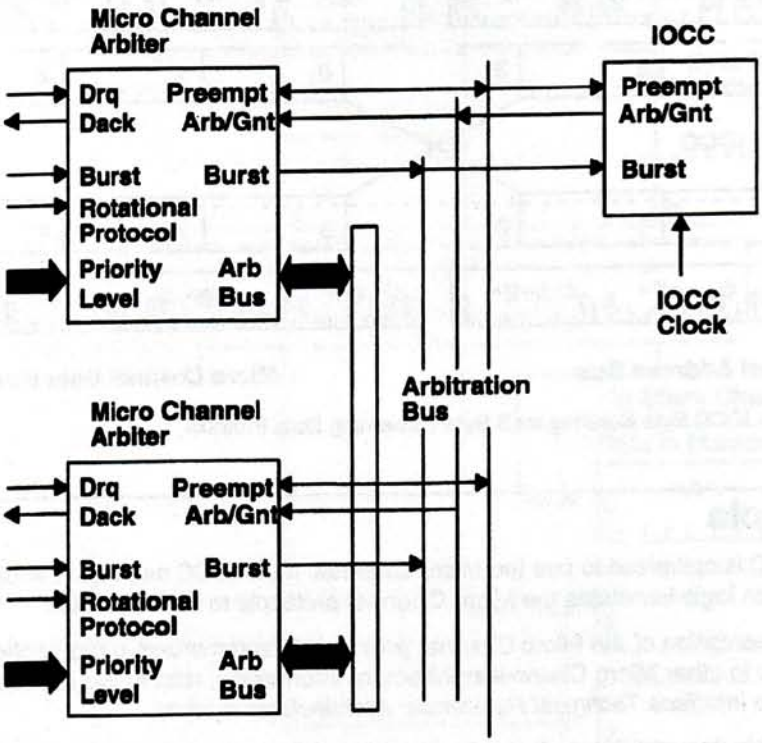


Figure 42. I/O Bus Arbitration

Figure 43 on page 2-15 shows an arbitration cycle. Devices request service by activating the 'preempt' signal. The IOCC responds by deactivating the 'arb/gnt' signal when the current bus owner completes its bus activity. Each requesting arbiter then presents its arbitration level on the arbitration bus. The IOCC then reactivates the 'arb/gnt' signal. If the device sees its arbitration level value on the arbitration bus, the device knows it has been granted use of the bus. Device Request (Drq) is a signal (internal to each of the device arbiters) that signals a request to arbitrate for the bus. Device Acknowledge (Dack) is a signal (internal to each of the device arbiters) that signals acknowledgement of being granted the bus.

Note: In some implementation, the arbitration bus might be multiple buses to the arbitration control logic, but the bus can be viewed as one logical bus from the device's perspective.

At the end of the bus cycle, the arbitration cycle is repeated if the 'burst' signal is not active. If there are no requesters, control is returned to the default arbiter at the arbitration bus level X'F'.

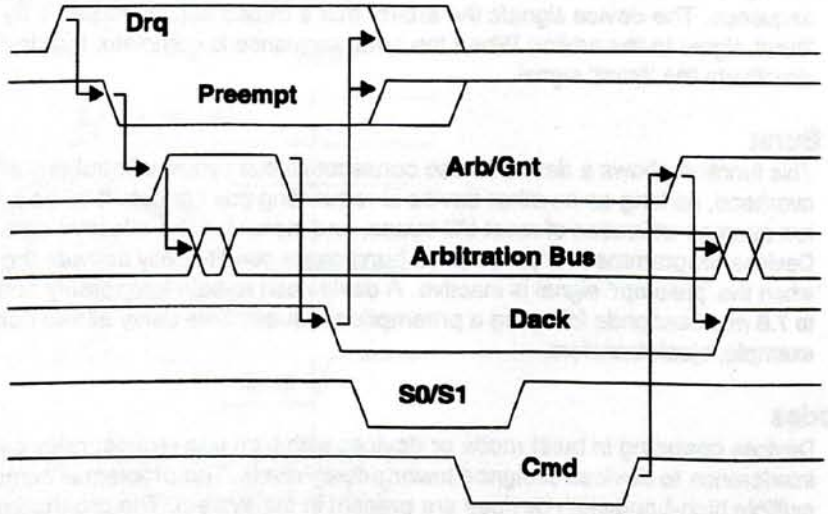


Figure 43. Arbitration Cycle

Both DMA slave and bus master devices utilize the arbitration mechanism to initiate bus cycles. The difference is that once granted use of the bus, the bus master device controls bus cycles, while the IOCC controls the bus cycles for DMA slave devices.

Priority Assignment

At startup, each device supporting arbitration is assigned a unique priority level ranging from X'0-F'. This priority level establishes the selection criteria to be used when contention exists. If multiple requests occur simultaneously, the device with the lowest numbered priority level is awarded use of the bus.

Arbitration level X'F' is always assigned to the system processor. If there are no other bus requesters, bus ownership defaults to level X'F'. Thus, the IOCC *owns* the I/O bus during idle conditions. Since I/O bus utilization is normally low, the IOCC does not normally arbitrate for the bus for I/O Load and Store instructions. Some IOCC implementations execute any pending I/O Load or Store instruction during the arbitration cycle (that is, when the 'arb/gnt' signal is in the 'arb' state), and extend the arbitration cycle as needed to complete the I/O Load or Store (up to the maximum time specified in the burst control field of the IOCC Configuration register). See "Implementation Details" on page 2-86.

Micro Channel I/O devices with long bursting characteristics should be designed using the Fairness (rotational) Arbitration Protocol, without which it is possible to lock out system processor I/O Load or Store instructions until the I/O device transfer is complete. If a lockout occurs for an extended period of time, a bus timeout error is posted, the 'arb/gnt' signal is set to the 'arb' state, and the 'reset' signals are activated to all slots. While the bus timeout error is active, all system processor I/O Load and Store instructions are guaranteed access to the bus.

NonPreemptive Burst

Devices can force nonpreemptive burst operations if it is necessary to retain control of the bus for short periods of time. Examples include use of a read-modify-write sequence in setting locks and use of a burst to allow the completion of a word-organized transfer sequence. The device signals the arbiter that a forced burst is required by activating the 'burst' signal to the arbiter. When the burst sequence is complete, the device must deactivate the 'burst' signal.

Preemptive Burst

This function allows a device to use consecutive bus cycles without any arbitration overhead, as long as no other device is requesting bus service. It takes advantage of the low average utilization of most I/O buses, and increases the effective data rate of a device. Devices programmed for preemptive burst mode conditionally activate the 'burst' signal when the 'preempt' signal is inactive. A device can remain temporarily nonpreemptive for up to 7.8 microseconds following a preemption request. This delay allows completion of, for example, block transfers.

Fairness Modes

Devices operating in burst mode or devices with high bus request rates can cause severe interference to devices assigned lower priority levels. The problem is compounded when multiple high-bandwidth devices are present in the system. The programmable *fairness* mode makes these high-bandwidth devices subject to preemption by any device. If multiple high-bandwidth devices are active simultaneously, service is rotated in a priority sequence, and each receives a percentage of bus cycles inversely proportional to the number of active bus requesters.

To meet wide variations in device operating requirements, arbiters are programmable to operate in either linear or fairness mode. Operating modes can be mixed on the same bus. Linear priority mode is provided to meet low latency requirements of unbuffered devices, while fairness mode provides a more equitable distribution of bus cycles in a high-demand environment, for example, with two or more high-bandwidth bus masters.

Fairness mode is a special case of preemptive burst. If there is only one bus requester, the current bus owner can utilize all of the bus bandwidth. As with preemptive burst, a device programmed in fairness mode can remain temporarily nonpreemptive for up to 7.8 microseconds following a preemption request.

DMA Slave Selection

The Micro Channel architecture allows a DMA slave to be selected either by its arbitration level or, optionally, by its I/O address (but not both). In these systems, the method supported for selection of DMA slave devices is by its arbitration level, status ('s0' exclusive-ored with 's1'), and an I/O cycle ('m/io' signal in the IO state).

Basic Transfer Cycle

Although the I/O architecture defined in this chapter is generic and allows the attachment of a number of unique buses, the intended design point is the Micro Channel bus. These bus protocols are shown in Figure 44.

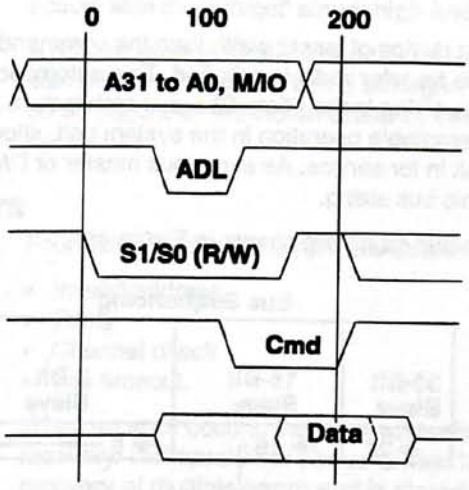


Figure 44. I/O Bus Cycles

The Micro Channel offers a 32-bit data path with 4G bytes of address space. It includes extensive support for reliability, availability, serviceability, extendibility, and configurability. The physical package and connector are designed to improve electrical characteristics.

Two status lines, 's0' and 's1', define the initiation of bus write and read cycles respectively, while the 'm/io' line differentiates between I/O memory and I/O devices. All addresses for the next cycle are overlapped with the processing of the current cycle. The bus architecture includes a special protocol for transferring sequential blocks of data. This is known as the Streaming Data protocol, and is described in the next section.

Streaming Data

The Streaming Data protocol is a single-address, multiple-data protocol that improves bus efficiency by amortizing bus-cycle arbitration and address setup across multiple data cycles. It has particular value in transferring data between a memory and a processor cache or between a memory and a high-performance I/O device.

Streaming data begins with a cycle similar to a standard basic transfer cycle, but switches to a clock synchronous transfer protocol.

Streaming data operations are supported for all IOCC transactions including Load and Store instructions, DMA slave, and bus master operations.

Following the activation of the 'cmd' signal, the bus master indicates Streaming Data Protocol capability by starting a bus clock called the 'sd strobe' signal. This clock is used by both the bus master and slave to clock data onto and off of the bus. As the operation proceeds, new data is placed on the bus every time the 'sd strobe' signal makes a high-to-low transition. For additional information on the Streaming Data Protocol, refer to "Implementation Details" on page 2-86 for system implementation specific information. For other Micro Channel architecture information, refer to the *Personal System/2 Hardware Interface Technical Reference: Architectures.*

Dynamic Bus Sizing

I/O bus read or write operations do not necessarily have to match the physical width of the device. The Micro Channel architecture requires that the current bus master automatically manage discrepancies in data transfer widths. The IOCC is considered to be the current bus master for processor initiated I/O Load and Store instructions, and thus, must manage logical data-width transformations.

A Load or Store instruction issued to a device of lesser width than the command causes multiple I/O cycles to be taken until the transfer width is satisfied. This automatic data-width matching is referred to as dynamic bus sizing in the Micro Channel architecture. The multiple I/O cycles complete as a preemptable operation in the system unit, allowing bus master and DMA slave cycles to break in for service. As such, bus master or DMA slave latency is unaffected by use of dynamic bus sizing.

Protocols and sequencing of dynamic bus sizing are shown in Figure 45.

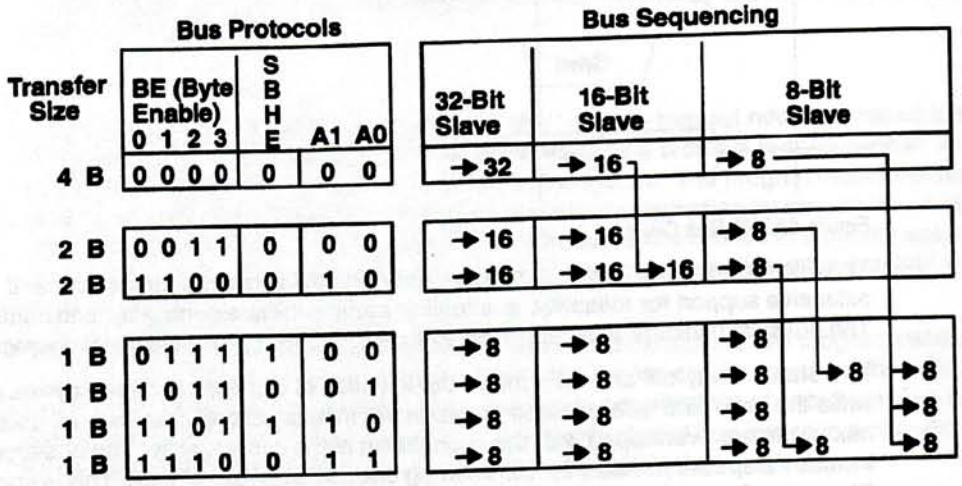


Figure 45. Dynamic Bus Sizing

It is generally recommended that the programmer writing an I/O device driver be aware of the physical characteristics of the target device. One should be aware when dynamic bus sizing is invoked by IOCC hardware since this operation requires more time to complete. See "String Operations" on page 2-28 for details on where this could be a problem.

Partial Transfer Cycles

Partial write operations (for example, writing one byte of a 2-byte device, or two bytes of a 4-byte device) are permitted in the bus architecture. The operations are useful in performing unaligned moves. The Micro Channel supports partial write operations when operating with both memory and I/O devices.

Bus write operations issued on address boundaries matching the device width allow completion of the operation in the minimum number of bus cycles. Operations issued to nonaligned addresses transfer the data to the device using multiple (partial write) cycles. These write operations use the bus 'sbhe'/'a0' and 'be0 to be3' protocols to write the desired portion of the word. Partial transfers apply to I/O Load and Store instructions and (potentially) to bus master and DMA slave operations when operating with bus memory.

Partial transfers can take two to four times the normal number of bus cycles and caution should be exercised in their use. If nonaligned, I/O Load and Store instructions halt the processor for a longer period of time, adding latency to system interrupt service. See "String Operations" on page 2-28 for details on where this could be a problem.

Bus Refresh

Bus refresh cycles are provided as a convenience to I/O devices with embedded random access memory (RAM). Refresh cycles occur at one of several periodic rates selectable by the Configuration register. Refer to "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86 for a description of refresh options. The refresh cycle occurs with the 'arb/gnt' signal high and does not consume a bus arbitration level.

A refresh cycle is similar to an I/O memory read operation, except that the 'refresh' line is also activated. Address bits 0 through 11 (using the Micro Channel notation shown in Figure 36 on page 2-8) are incremented by one, and are placed on the bus during the refresh cycle.

Bus Errors

Four different kinds of errors are detectable on the Micro Channel:

- Invalid address
- Parity
- Channel check
- Bus timeout.

When an error occurs, the error status is logged in IOCC registers as an aid in error recovery. Individual error status is kept for each I/O device (by arbitration level) to assist in recovery of multiple errors and is stored in the Channel Status register associated with that device. I/O Load and Store instructions utilize channel 15 in regular operation and error status for those operations is saved in that set of registers. Refer to "Load and Store Error Conditions" on page 2-31 for a description of this error status.

Invalid Address

The Micro Channel architecture requires a positive response to all addresses. Address response is signalled on the Micro Channel by driving the 'cd sfdbk' signal low. Failure to respond indicates that the address is invalid, or is issued to a missing or mis-seated card.

If an I/O Load or Store instruction is issued with Segment Register bit 12 on, the IOCC checks for this address response. If none is received, a Data Storage Interrupt (DSI) is issued and a card selected feedback error code is set in Channel Status register 15. Refer to "I/O Segment Register Definition" on page 2-26 for additional details.

Parity Errors

The Micro Channel architecture definition includes address and data parity functions. Checking is performed only when both the bus master and slave support parity. Refer to "Exception Reporting and Handling" on page 2-85 for details of the I/O parity support.

Channel Check

The Micro Channel includes a 'chck' signal that indicates an unusual event occurred during the bus cycle. Examples include data parity error and page fault.

For details on the use of the 'chck' signal in reporting exception conditions within the unit, see "Exception Reporting and Handling" on page 2-85.

It is important to note that the unit is designed to recover from synchronous channel checks. Adapters that use the 'chck' signal asynchronously make an Initial Program Load (IPL), the only recovery that is possible.

Bus Time Out

A number of conditions can result in a *hung* bus or in grossly extended I/O bus cycles. These errors can result in overrun conditions to other devices on the I/O bus and are checked by the IOCC using a bus timeout mechanism. Although the minimum architected bus timeout value is 7.8 microseconds, the IOCC does not attempt to check that finely and should implement a timeout that varies between 15 and 120 microseconds. See "Implementation Details" on page 2-86.

Bus hang problems are caused by either hardware or software errors. These errors are generally associated with arbitration for the I/O bus followed by failure to complete the bus cycle.

On a bus timeout error, the IOCC deactivates the 'arb/gnt' signal, and sets bit 1 (the bus timeout bit) in the IOCC Miscellaneous Interrupt register, and generates an interrupt. This error is considered to be uncorrectable and the master enable control in the IOCC Configuration register is reset. This disables all interrupt and channel requests. Also, a 'reset' signal is applied to all I/O slots. In addition, if an I/O Load or Store instruction is pending in the IOCC when the bus timeout occurs, and the target of that Load or Store instruction is the Micro Channel bus, then a Data Storage interrupt is sent for the terminated Load or Store Instruction. If an I/O Load or Store instruction is pending in the IOCC when the bus timeout occurs and the target of that Load or Store instruction is an IOCC facility, then the load or store instruction will be completed after the Micro Channel bus is cleared by the IOCC. The IOCC internal status is unchanged, so that channel conditions at the time of the error can be logged. As an aid in determining the cause of the error, extraneous bus status is also captured in the Bus Status register.

Incorrect programming of the DMA controller can result in a hung bus. The DMA controller includes multiple channels; each can be personalized to control either a bus master or DMA slave device. Personalization can be dynamically performed. If a programmer should personalize a channel for bus master operation, but the device is actually a DMA slave device, the bus will hang on the first DMA request that the device makes.

Interrupt

Eleven Micro Channel interrupt lines are supported by the IOCC. Interrupts on the Micro Channel are level-sensitive, active-low, and exhibit natural interrupt-sharing capabilities. The I/O Board provides pull-up resistors on all Micro Channel interrupt signals so that unused lines float to the inactive state. Refer to "I/O Interrupts" on page 2-68 for additional details.

Programming Model

The following section describes the programming model for the I/O bus support functions provided by the IOCC.

Load and Store Instructions

The Load and Store instructions can be issued to devices on the I/O bus in a similar manner that they are issued to system memory. The programmer specifies a Segment register identifying a specific address space and supplies an offset into that space. The offset is obtained from the effective address and is not translated prior to being applied as a bus address. Figure 46 shows the process.

I/O Load and Store instructions are under control of the Segment registers. A command is directed to the I/O bus when the type (T) bit of the Segment register is set to a value of 1 and the bus unit id (BUID) address is set to select the IOCC. Some I/O operations require that the privileged key (K) be set to a value of 0 (the privileged mode).

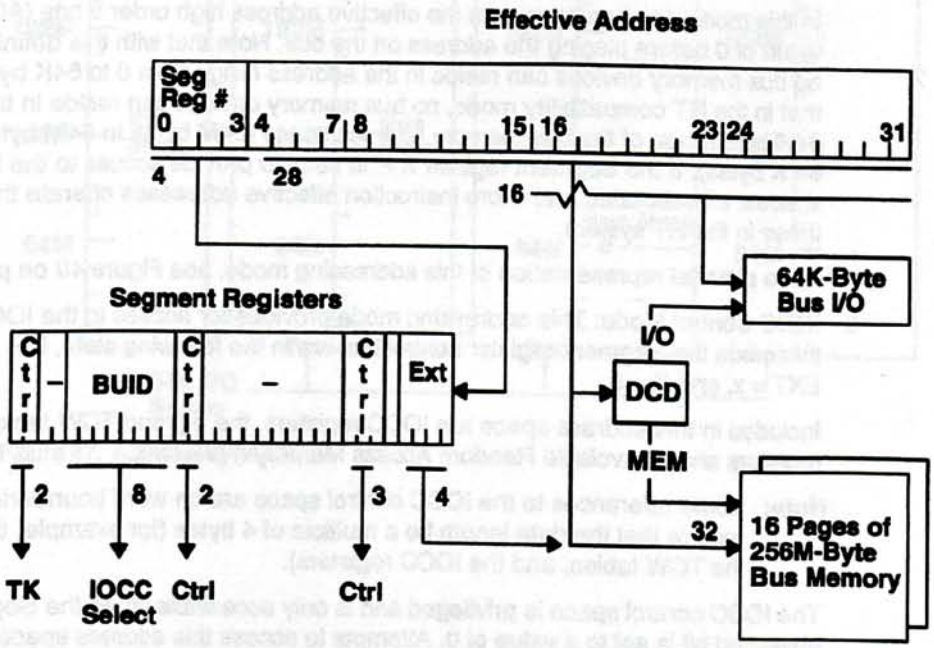


Figure 46. I/O Addressing

Address Spaces and Effective Addresses

Figure 47 on page 2-23 illustrates the addressing modes. I/O addressing requirements are met by having multiple address spaces. These address spaces are selected by way of control bits in the Segment register (see Figure 51 on page 2-26) resulting in three I/O effective address operating modes as follows:

1. Standard Bus Mode: This I/O effective address mode provides for 32-bit addressing of the I/O bus. In this mode the Segment register control bits are in the following state: T = 1, I = 0, and M = 0.

   The 32 bit bus memory address is formed by concatenating 28 bits of the effective address with the 4 extent (EXT) bits from the Segment register. This partitions the bus memory device space into 16 pages of 256M bytes each (4G bytes of total address space), and separate Segment registers must be used to address adjacent 256 M-byte

address spaces. The 16 bit I/O device address is taken directly from the effective address. To address a device within the 64K byte Micro Channel I/O space, effective address bits 4 through 15 and Segment register bits 28 through 31 must all be set to a value of 0. Effective addresses are not translated, but are used as *real* addresses into the I/O space.

For a pictorial representation of this addressing mode, see Figure 48 on page 2-24.

2. RT Compatibility Mode: This addressing mode assists in the simulation of the RT system allowing for 24 bit addressing. In this mode the Segment register control bits are in the following state: T = 1, I = 0, M = 1, and EXT = x.

   In this mode, 16M bytes of bus memory is selected using an effective address of X'x4 xxxx xx', and 64K bytes of bus I/O using X'x0 00 xx xx'. Any other effective addressing range other than these two results in a Data Storage interrupt and an invalid operation error status is set in the Channel Status register (CSR) 15. This mode maintains compatibility with the I/O structure of the RT system and provides the ability to replace an RT object code Load or Store instruction with its system equivalent, and the simulator does not have to worry about differences in the effective address format.

   In this mode, the hardware sets the effective address high order 8 bits (A0 to A7) to a value of 0 before placing the address on the bus. Note that with this definition of the bus, no bus memory devices can reside in the address range from 0 to 64K bytes. Also note that in the RT compatibility mode, no bus memory devices can reside in the lower 64K-byte range of the bus memory address space (64M bytes to 64M bytes + 64 K bytes). If the Segment register X'F' is used to provide access to the IOCC address spaces, all user Load and Store instruction effective addresses operate the same as those in the RT system.

   For a pictorial representation of this addressing mode, see Figure 49 on page 2-24.

3. IOCC Control Mode: This addressing mode provides for access to the IOCC facilities. In this mode the Segment register control bits are in the following state, T = 1, I = 1, M = x, EXT = x, and K = 0.

   Included in this address space are IOCC registers, the tag and TCW tables, the System registers and Nonvolatile Random Access Memory (NVRAM).

   Note: Some references to the IOCC control space are on word boundaries only and require that the data length be a multiple of 4 bytes (for example, the tag tables, the TCW tables, and the IOCC registers).

   The IOCC control space is privileged and is only accessible when the Segment register privileged bit is set to a value of 0. Attempts to access this address space when the Segment register privileged bit is set to a value of 1 causes a Data Storage interrupt to be posted and and invalid operation error status to be set in the Channel Status register 15. Attempts to access undefined effective addresses in the IOCC control address space also results in a Data Storage interrupt (invalid operation).

   For a pictorial representation of this addressing mode, see Figure 50 on page 2-25.

Although bus memory and bus I/O are disjointed in PC products, the system unit maps these two address spaces together. Since bus I/O only requires 64K bytes of addressing, this address space easily maps into the low addresses of the (4G bytes) bus memory address space. The architecture of PC products is such that no bus memory feature cards may be hardwired in the address range of 0 to 64K bytes, and no address conflicts exist. Note that the 64K bytes of Micro Channel I/O space can be accessed when utilizing each of the three effective address operating modes as shown in Figure 47. The values for the T, I and M bits for each of the three I/O effective address operating modes were previously described and are illustrated in Figure 47.

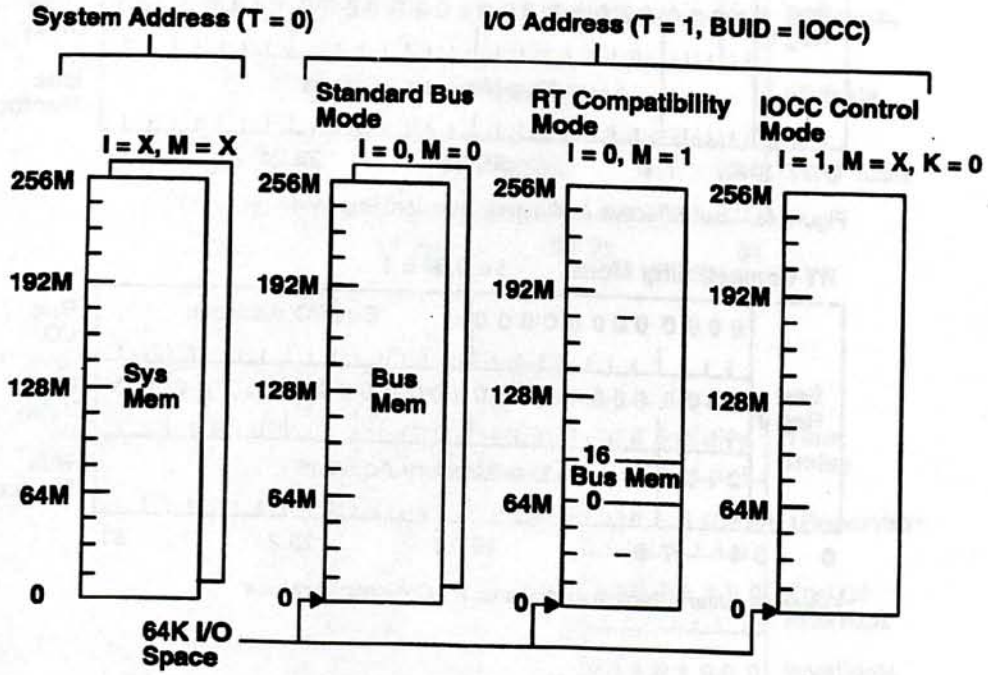


Figure 47. Addressing Model

Figures 48, 49, and 50 summarize the system effective addresses. Effective addresses are obtained from the processor general purpose registers and are under user control. If a bus memory page is mapped to system memory, the bus address is translated to the address of the mapped system memory page.

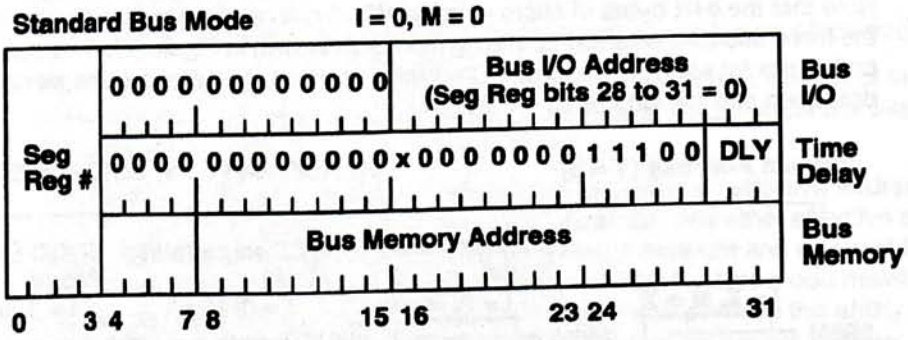


Figure 48. User Effective Addresses: Standard Bus Mode

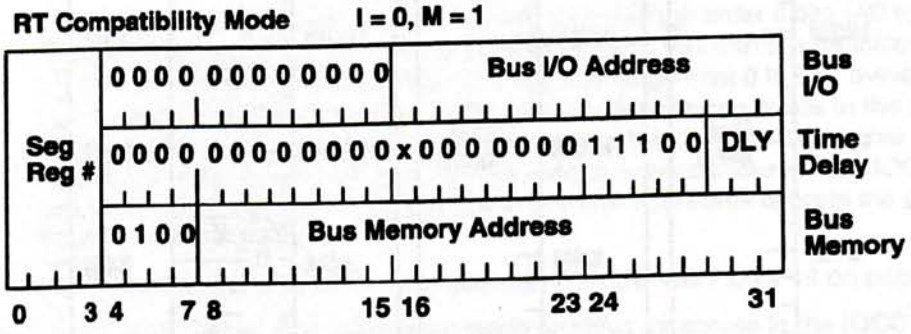


Figure 49. User Effective Addresses: RT Compatibility Mode

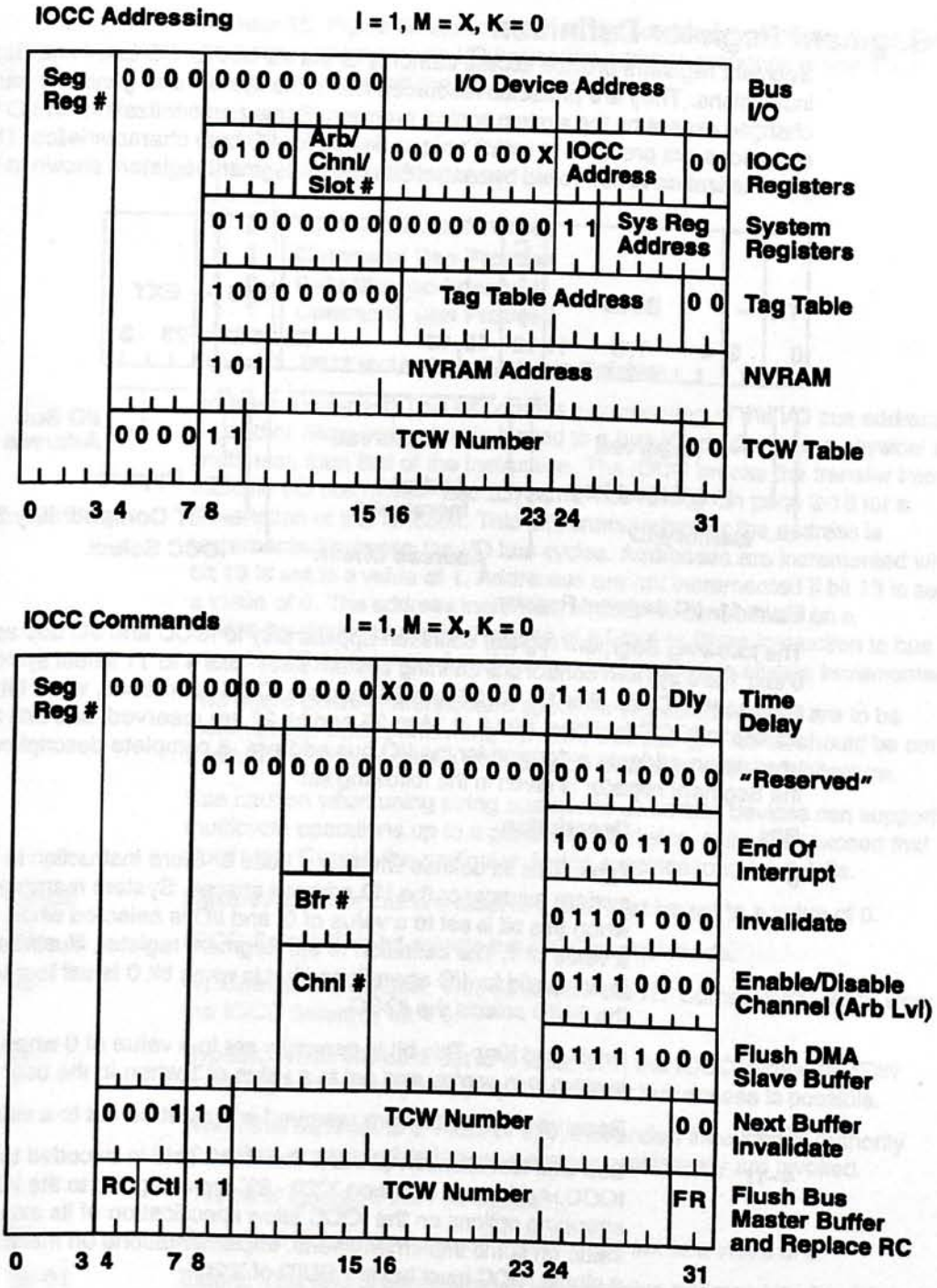


Figure 50. IOCC Effective Addresses

I/O Segment Register Definition

Segment registers provide access authority to the I/O bus for I/O Load and Store instructions. They are protected resources within the system and generally cannot be changed except by the system control program. Some personalizations of I/O bus operations are provided to match unique device (or I/O bus) characteristics. This personalization is controlled by control bits in the Segment registers shown in Figure 51.

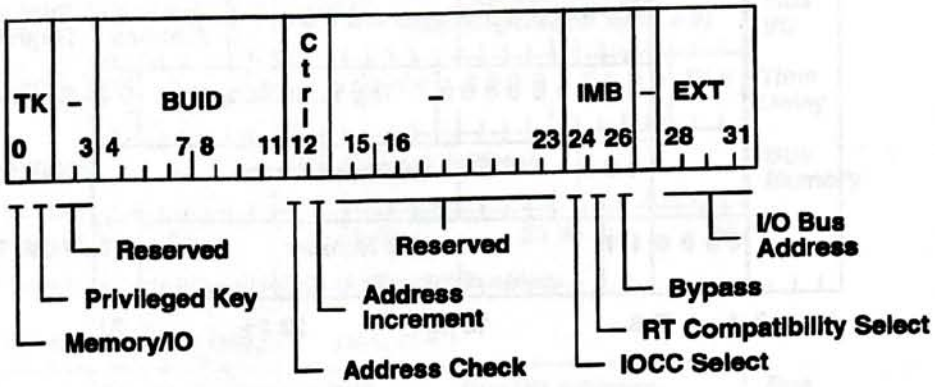


Figure 51. I/O Segment Register

The following Segment register definition applies only to IOCC and I/O bus applications. Bits 0 and 1 are system control bits defining system state. Bits 4 to 11 select system facilities such as the IOCC. Bits 12, 13, 25 and 26 mediate IOCC operations, while bit 24 provides access to IOCC facilities. Bits 2, 3,14 to 23 and bit 27 are reserved, and bits 28 to 31 are used as an address extension for the I/O bus address. A complete description of all fields in the Segment register is given in the following list:

Bits	Description
0	Type: This bit defines whether a Load or Store instruction is targeted to system memory or the I/O address spaces. System memory is selected when this bit is set to a value of 0, and I/O is selected when this bit is set to a value of 1. The definition of the Segment register, illustrated in Figure 51, is only valid for I/O operations, that is when bit 0 is set to a value of 1, and the BUID selects the IOCC.
1	Privileged Key: This bit is generally set to a value of 0 when the operating system is in control and set to a value of 1 when in the user mode.
2–3	Reserved: These bits are reserved and must be set to a value of 0.
4–11	Bus Unit Identification (BUID): The BUID field is decoded to select the IOCC. Addresses between X'20 - 23' are assigned to the IOCC. Hardware strapping options on the IOCC allow specification of its exact BUID field value on some implementations. Implementations on machines that support a single IOCC must have a BUID of X'20'.
12	Address Check: This bit provides for conditional checking of I/O addresses during Load and Store instructions. The Micro Channel provides for a positive address response by device activation of the 'cd sfdbk' line. If this line is not activated, the device address is invalid. See "Invalid Address" on page 2-19. An I/O Load or Store instruction that does not receive a positive address response is allowed to proceed when bit 12 in the Segment register is set to a value of 0. A command issued to an invalid device address when bit 12 is set to a value of 1 causes a Data Storage interrupt to be posted and a card selected feedback error code to be set in Channel Status

register 15. Figure 52 summarizes all the combinations of bit 12 and the address response by an I/O board (the address response is true if the device has activated the 'cd sfdbk' line).

Bit 12	Address Response	
0	0	Command Can Proceed
0	1	Command Can Proceed
1	0	Data Storage Interrupt
1	1	Command Can Proceed

Figure 52. Bit 12 and Address Response Definition

13 Address Increment: This bit controls incrementing of the I/O bus address if a Load or Store instruction is issued to a bus I/O device with a physical data width less than that of the instruction. The IOCC breaks the transfer into multiple I/O bus cycles. See "Dynamic Bus Sizing" on page 2-18 for a description of this function. This bit controls whether the address is incremented between the I/O bus cycles. Addresses are incremented when bit 13 is set to a value of 1. Addresses are *not* incremented if bit 13 is set to a value of 0. The address increment function is controllable on a device-by-device basis. In the case of a Load or Store instruction to bus memory, bit 13 is ignored and the bus addresses are always incremented.

The Micro Channel architecture specifies that all addresses are to be incremented when performing dynamic bus sizing. This bit should be set to a value of 1 when working with devices designed to this architecture.

Use caution when using string operations as certain devices can support multicycle operations up to a particular word size, but not to exceed that word size. Consult the particular device specifications for details.

14–23 Reserved: These bits are reserved and must be set to a value of 0.

24 IOCC Select: This bit selects the IOCC control mode.

25 RT Compatibility Select: This bit selects the RT Compatibility Mode when the IOCC Select (I) bit = 0.

26 Bypass: When this bit is set to a value of 1, the IOCC bypasses TCW checking and memory mapping. Only direct bus access is possible.

When this bit is set to a value of 0, the extended functions of authority checking, access validation, and system consistency are invoked.

This bit is ignored if the I bit equals 1.

27 Reserved: This bit is reserved and must be set to a value of 0.

28–31 Extent: This field is concatenated with effective address bits 4 to 31, to form a 32-bit I/O bus address when working in standard bus mode. It is gated to address bits 'A31' to 'A28' on the I/O bus.

Address and Data Alignment

Data for Load and Store instructions is normally right-justified in the Processor register. One-byte operands are located in byte 3. Two-byte operands are located in bytes 2 and 3. String operations are an exception and are left-justified in the starting Processor register.

Target I/O device addresses should normally be aligned on boundaries equal to the device width. This maintains optimal performance when performing Load and Store instructions. If this rule is not observed, the IOCC performs the operation using multiple (narrower) I/O bus cycles. This can take up to four times longer to complete the Load or Store operation. Refer to "Partial Transfer Cycles" on page 2-18 for additional details.

String Operations

String operations allow the issuance of Load or Store instructions with data widths from 1 to 128 bytes. The bus protocol used in the data transfer is dependent on the I/O device. String operations are applicable to any addressable device on the Micro Channel and to the tag tables, TCW tables, and to the NVRAM within the IOCC address space. However, for some I/O devices, applicability of string operations may be limited by the device itself.

String operations issued to normal PC devices are performed using standard bus protocols. Multiple bus cycles are issued, using dynamic bus sizing, until the transfer length is satisfied. These multiple cycles operate under preemptive burst arbitration rules and Load or Store string instructions are momentarily suspended if any I/O device requests DMA slave or bus master operation.

String operations issued to devices supporting the streaming data transfer protocol use that protocol where appropriate. This protocol operates under non-preemptive burst arbitration rules. In the case of string operations, however, the amount of time from the preempt request by a device until the IOCC releases the bus is controlled by the Burst Control bits in the IOCC Configuration register (see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86).

It is generally recommended that the programmer writing an I/O device driver be aware of the physical characteristics of the target device when using string operations. One should be aware of the effects of dynamic bus sizing and partial transfers, since these operations require more time to complete. Refer to "Dynamic Bus Sizing" on page 2-18 and "Partial Transfer Cycles" on page 2-18 for details of these functions. Slower than expected I/O instruction processing can have detrimental effects on system performance. For example, the system processor cannot accept an interrupt while I/O Load or Store instructions are in process. Both dynamic bus sizing and unaligned moves (partial transfers) take longer to complete, adding latency to system interrupt service. Although most devices are reasonably fast and do not cause any problems, this latency can be large if extended string operations are performed against slow devices.

Load and Store Access Authority Checking

I/O Load and Store instructions are subject to access authority checking. Separate mechanisms are used for checking bus I/O and bus memory, as shown in Figure 53. Bus I/O accesses are checked by way of a base and bounds (range) check, while memory accesses are verified by way of a storage key in the TCW table. If the page is mapped to system memory, write authority is also verified. Load and Store instructions to bus memory or (shared) system memory are treated like a bus master operating on channel 15 and use IOCC registers associated with that channel.

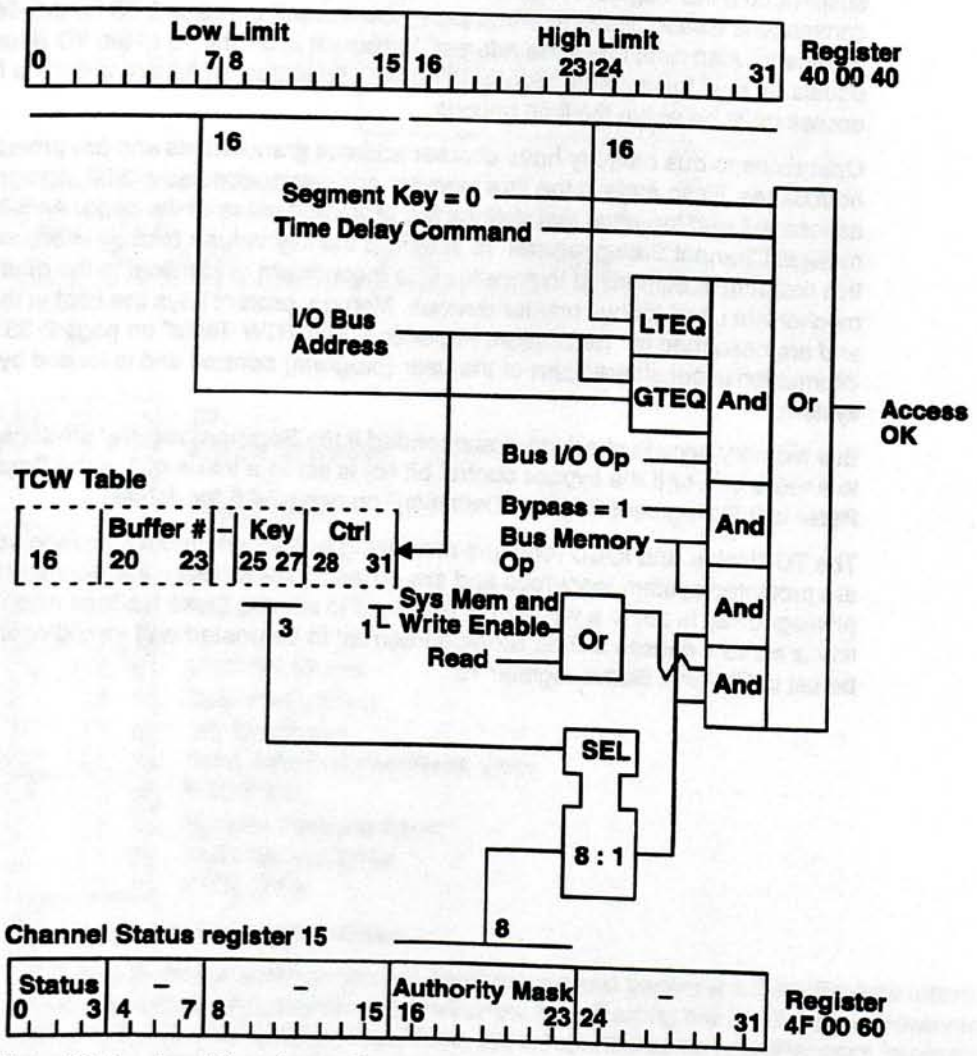


Figure 53. Load and Store Access Authority Checking

Operations to bus I/O have fine address granularities. The operations are verified by way of address range checking. Address ranges are controlled by the operating system and restrict access of user programs to authorized devices. Address range information is considered part of the user (program) context and is loaded into an IOCC register by the operating system. This register defines a contiguous range of authorized I/O addresses with a minimum address granularity of 1 byte. Invalid access attempts cause a Data Storage interrupt to be posted and a limit check error code to be set in Channel Status register 15. This interrupt is precise for all I/O Load and Store instructions. Address range checking is suspended if the Segment register privileged key is set to a value of 0, or if a **time delay** command is issued. Refer to "Time Delay Command" on page 2-63 for details of this command. Also note that if the address increment is off (bit 13 of the I/O Segment register equals 0), only the starting address is tested. If address increment is on, the full length of the access must be within the limit bounds.

Operations to bus memory have coarser address granularities and are protected on page boundaries. Each page in the bus memory address space has a 3-bit storage protect key associated with the page that defines the protection class of the page. An 8-bit authority mask in Channel Status register 15 specifies the key values (and by inference, pages) that this program is authorized to access. This mechanism is identical to the memory protect mechanism used for bus master devices. Memory protect keys are kept in the TCW table and are described in "Translation, Protection, and TCW Table" on page 2-33. The mask information is considered part of the user (program) context and is loaded by the operating system.

Bus memory access checking is suspended if the Segment register privileged key (K) is set to a value of 0 or if the bypass control bit (B) is set to a value of 1 in the Segment register. Refer to "I/O Segment Register Definition" on page 2-26 for details.

The TCW table and IOCC registers containing limit check information and authority masks are protected system resources and are only accessible when the Segment register privileged key is set to a value of 0. Attempts to access these facilities when the privileged key is set to 1 causes a Data Storage interrupt to be posted and invalid operation status to be set in Channel Status register 15.

Load and Store Error Conditions

Error conditions that arise in Load and Store instructions include bus errors, programming errors, and hardware errors. If no previous error remains in the Channel Status register 15, then the specific cause of the error is placed into the Channel Status register 15 bits 0 to 3. The IOCC only places the first error code into Channel Status register 15. Figure 54 shows the resultant register contents.

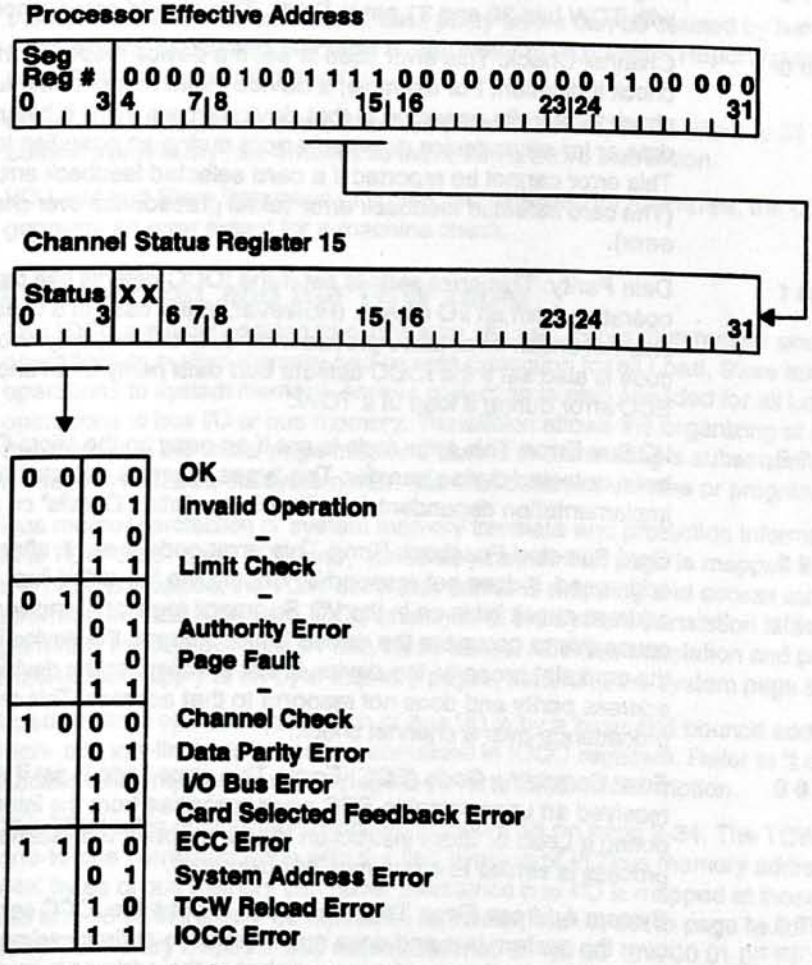


Figure 54. Load and Store Error Status

Load and Store instruction errors are synchronous and generate a Data Storage interrupt. No device should asynchronously report errors by activating the 'chck' signal. However, if this occurs, the error is not reported here, but is reported as an miscellaneous interrupt as described in "I/O Interrupts" on page 2-68. Refer to "Exception Reporting and Handling" on page 2-85 for more information. Load and store error codes are summarized as follows:

Error Code	Description
0 0 0 1	Invalid Operation: This error code is set if an attempt is made to access a facility or device not authorized by the system supervisor. It is also set if an attempt is made to access a bus address for which a TCW does not exist (except when the bypass bit is on).
0 0 1 1	Limit Check: This error code is set if an attempt is made to access a bus I/O device not within the address range established by the limit registers.

Error Code	Description
0 1 0 1	Authority Error: This error code is set if an attempt is made to access a bus or system memory page and the storage key in the TCW does not match the authority mask in Channel Status register 15. It can also be set if a write operation is attempted to a read-only page in system memory.
0 1 1 0	Page Fault: This error code is set if an attempt is made to access a page with TCW bits 30 and 31 set to B'01'. This should occur in normal operation.
1 0 0 0	Channel Check: This error code is set if a device responds with a channel check indication. For example, a device might respond with a channel check for a write operation to that device where there is bad parity on the data or for other device detected errors during an operation to that device. This error cannot be reported if a card selected feedback error is reported. (The card selected feedback error takes precedence over channel check error).
1 0 0 1	Data Parity: This error code is set if the IOCC detects bad parity on a Load operation from an I/O device. (However, in the case of a Load operation, a channel check error takes precedence over a data parity error.) This error code is also set if the IOCC detects bad data parity or an uncorrectable ECC error during a load of a TCW.
1 0 1 0	I/O Bus Error: This error code is set if an error on the Micro Channel has been detected during transfer. The types of errors detected here are implementation dependent (see "Implementation Details" on page 2-86).
1 0 1 1	Card Selected Feedback Error: This error code is set if, after a device is addressed, it does not respond by driving the 'cd sfdbk' line, and the address check bit is on in the I/O Segment register. Conditions which could cause this to occur are the device is not present, the device is not seated in the card slot properly, the device is not enabled, or the device detects bad address parity and does not respond to that address. This error code takes precedence over a channel check.
1 1 0 0	Error Correcting Code (ECC) Error: This error code is set if the IOCC received an uncorrectable ECC error response from the internal system bus during a Load or Store instruction that is mapped to system memory. (This process is similar to a bus master operation).
1 1 0 1	System Address Error: This error code is set if the IOCC sends an address over the system bus and does not receive an address acknowledgement. This can occur if the real page number in the address is invalid. Software should make sure that the real page number in the TCW is valid.
1 1 1 0	TCW Reload Error: This error code is set if the IOCC detects a parity or uncorrectable ECC error during an indirect TCW reload (with the bypass bit off).
1 1 1 1	IOCC Error: This error code is set if the IOCC detects an internal error during a Load or Store instruction. This error only occurs in a TCW and Tag table access or **flush** command. All other IOCC errors result in a check stop.

No provision is made to capture status for multiple errors. If this should occur, Channel Status register 15 contains error information relating to the first error. On some implementations, Channel Status register 15 bits 6 to 31 may be indeterminate after an error. Therefore, software should restore Channel Status register to a known state after an error.

Channel 15 always remains enabled following an error, or a deadlock situation would exist.

Synchronous errors are precise, and a retry may be attempted as part of the error recovery. Certain other errors associated with an I/O Load or Store instruction may not be synchronous, and are not reflected in this register. An example of these errors include delayed channel check response (see "Exception Reporting and Handling"" on page 2-85) and a bus timeout condition (see "Bus Timeout" on page 2-20 for more information).

I/O bus errors such as address or data parity errors can be caused by hardware malfunctions or transient electrical noise. Refer to "Exception Reporting and Handling" on page 2-85 for more information.

On a Load instruction, if bits 0-3 are all 0, the value of CSR15 bits 4 to 31 are whatever software previously had written into them with a Store instruction.

I/O Load and Store instructions to the IOCC facilities (for examples, the CSRs) do not generate an error except for a machine check.

Translation, Protection, and the TCW Table

The IOCC provides address translation for all Load, Store, bus master and DMA slave operations to system memory and access protection for all Load, Store and bus master operations to system memory. Access protection is also provided for all Load and Store operations to bus I/O or bus memory. Translation allows the organizing of I/O buffers within the context of the virtual page map and assists in eliminating a subsequent move operation. Protection insulates the system from non-well behaved devices or programs.

Bus memory protection or system memory translate and protection information is contained in a TCW table. Each TCW entry identifies whether that page is mapped to system memory. If a page is mapped, the TCW entry also contains mapping and access authority information. This table is an IOCC analogue of the system translation tables, and is generally managed in concert with those tables. Address translation and protection mechanisms apply to 4K-byte memory pages, matching the system page size.

Load or Store operation protection of bus I/O is by a base and bounds address check. The high- and low-limit addresses are contained in IOCC registers. Refer to "Load and Store Access Authority Checking" on page 2-29 for a detailed description.

The TCW table organization is shown in Figure 55 on page 2-34. The TCW table has a one-to-one correspondence with the first *n* pages of I/O bus memory addresses. The first 64K bytes of bus memory can never exist since bus I/O is mapped at those addresses, and the first 16 TCWs should be initialized as invalid, that is, set to page fault. Thus, the first valid TCW entry maps I/O bus addresses X'00 01 00 00' to X'00 01 0F FF'; the second entry controls mapping of addresses X'00 01 10 00' to X'00 01 1F FF', and so on.

The number of bus memory addresses that can be mapped depends on how much TCW Random Access Memory (RAM) is supplied by the IOCC. This amount is product dependent. A field in the IOCC Configuration register is used to specify the amount of TCW RAM supplied. Refer to "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86 for details. Access to the TCW table entries must be 4-byte aligned and must be an exact multiple of four bytes in length.

If the bus memory I/O address is mapped to system memory, the Real Page Number (RPN) in the TCW is used to access system memory. Otherwise, the address is directly applied to the I/O bus.

On a load instruction, if bits 0 to 3 all have a value of 0, the value of CSR 15 bits 4 to 31 is whatever software previously had written into them with a store instruction.

I/O load and store instructions to the IOCC facility (for example, the CSRs) does not generate an error except for a machine check.

The TCW table is a protected system resource located in the IOCC address space between addresses X'–*x* C0 00 00' and X'–*x* FF FF FF' (where *x* indicates any hexadecimal digit between 0 and F). It is only accessible to Load and Store instructions from the system processor when the Segment register privileged key is set to a value of 0. Attempts to access this table when the privileged key is set to a value of 1 causes a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

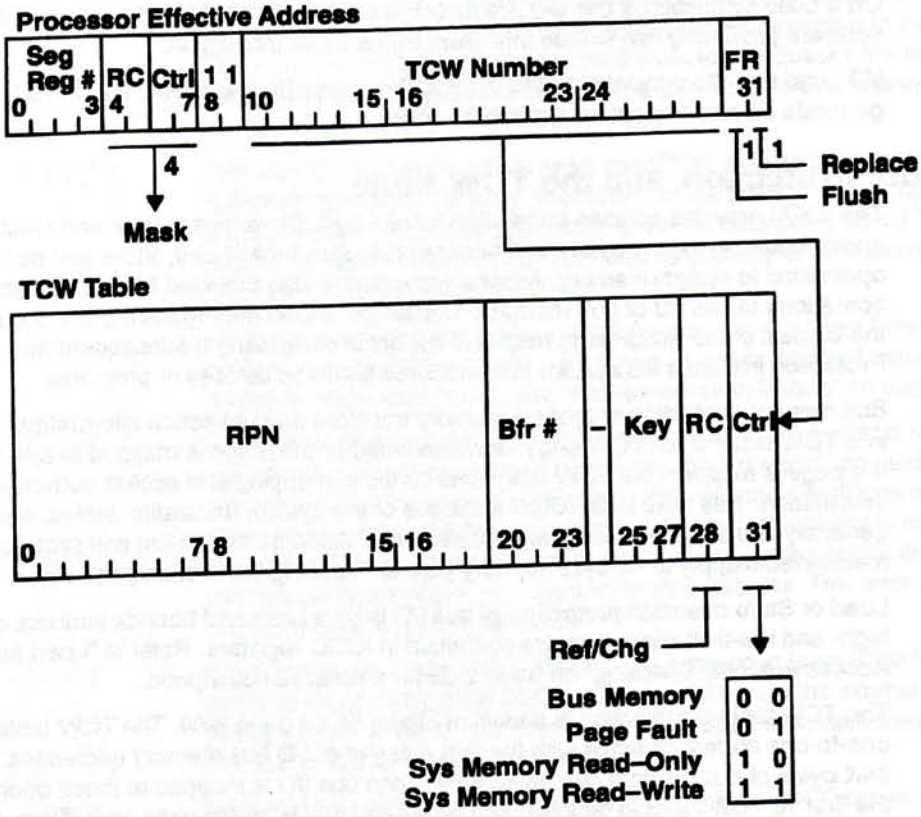


Figure 55. TCW Table

TCWs can be used for both bus master and DMA slave operations. A TCW entry is described in detail as follows. (Some fields described in the following section may be implementation-dependent as noted.)

Bits	Description
0–19	Real Page Number: This field in the TCW contains the real page address that the bus address is mapped to in system memory. Software should ensure that the RPN is valid (for example, is not outside the range of real memory).
20–23	Buffer Number: On buffered implementations, this field contains a 4-bit number specifying which of 16 buffers can be used by the IOCC when operating with this page. Although any buffer number may generally be assigned to any page, exercise caution since buffer sharing is not possible with DMA slave channels when tags are used. Personalization of a channel

for a DMA slave operation causes that channel to use the same buffer number. On implementations not buffered, these bits are indeterminate.

Note: Buffer number 'F' has some special restrictions and uses:

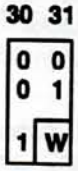
- Should always be used for Load and Stores instructions to bus memory when the bypass bit in the Segment register is off.
- Should never be assigned for any operations involving system memory (that is, where bit 30 of TCW word 0 is set to a value of 1).

24 Reserved and must be set to a value of 0.

25–27 Page Protect Key: This field contains a 3-bit key specifying the protection class of the page. Memory pages are assigned to one of eight protection classes. When a device initially arbitrates for the bus, an 8-bit access authority mask is obtained from the Channel Status register associated with that device. When a page is accessed, the key obtained from the TCW specifies the mask bit to be tested. If the selected bit is set to a value of 1, the access is permitted. Mask information for I/O Load and Store instructions are contained in Channel Status register 15. Load or store references to a bus memory page without the appropriate authority cause a Data Storage interrupt and set an access authority error code in Channel Status register 15. Refer to "Load and Store Access Authority Checking"" on page 2-29 for details. Similarly, invalid access attempts by a bus master device terminate the operation for this device and set an access authority error code in the Channel Status register associated with the device. Refer to "Bus Master Access Authority Checking" on page 2-46 for details.

28–29 Reference and Change (RC): These bits are equivalent to the RC bits in the system page frame table. Bus master transfers and shared memory Load and Store instructions do not modify the page frame table. As an aid in page management, the IOCC provides the reference and change history of all of its pages. This can be used to improve system performance in paging operations. Whenever a page is accessed, the IOCC sets its associated reference bit in the TCW table to a value of 1. Similarly, whenever a page is written, the IOCC sets both the reference and change bits to a value of 1. The B'01' code point is never naturally set by hardware and is only set by software to assist in page management. Note that these bits only apply to pages mapped to system memory.

30–31 Page Mapping and Control: These bits define page mapping and read-write authority. They are coded as shown in Figure 56.



Bus Memory
Page Fault (No Access)
System Memory

Figure 56. Page Mapping and Control Bits

Code points B'0X' signify that the page is not mapped to system memory. Code point B'00' should be set to allow accesses to memory devices on the I/O bus. Code point B'01' should be set when a page is not mapped and no device is present at that address. It causes a Data Storage interrupt if the operation is a load or a store, and a synchronous channel check response if the operation is a bus master transfer. Both of these actions are interpreted as an I/O bus page fault. Bus master devices designed to take advantage of this function are expected to halt and wait for the system to take corrective action.

Code point B'1X' signifies that the page is mapped to system memory. For Programmed I/O (PIO) operations, it causes the IOCC to redirect references to system memory using the TCW mechanism. Note that PIO to system memory using the TCW mechanism is implementation dependent. (See "Implementation Details" on page 2-86.) Bit 27 of the IOCC Configuration register is set at a value of 0 if PIO to system memory is supported. If not supported (bit 27 equals 1), a PIO Load and Store instruction results in a Data Storage interrupt.

Bus master operations are mapped by channel and enabled as defined by bits 2 and 3 of the status field of the Channel Status register. Note that bit 30 should match bit 2 of the status field of the Channel Status register; otherwise, it is treated as a page fault error condition as described in the preceding text.

Bit 31 controls write authority; if set to a value of 1, the page can be written. Note that the K bit (bit 1, or the Privileged bit) in the Segment register overrides bit 31, that is, privileged access is not limited by the Read-Write or Read-Only bit.

Maintaining Consistency

With various caches and buffers in a system, it is possible that the same data might exist in several places in the system. It then becomes the challenge of the hardware and software to maintain the consistency of these various copies. The I/O Architecture features that assist in maintaining consistency are the subject of this section.

Currently the I/O architecture defines two different modes of operation when it comes to Cache Buffer Support and Cache Coherency (as specified by two bits in the IOCC Configuration register). These are:

- Unbuffered Mode
- Buffered Mode.

Each of these modes has slightly different characteristics when it comes to keeping consistency among the various copies of the data in the system. These modes are discussed in the following sections.

Unbuffered Mode

In this mode, it is the responsibility of the hardware to keep everything consistent.

Buffered Mode

Figure 57 is a simplified version of Figure 35 on page 2-4, with only the data flow shown. Notice that there are (potentially) three copies of the same data in the system (shown as shaded boxes); one copy in the system memory, one copy in the processor data cache, and one copy in the IOCC buffer. In this mode, the software is responsible for keeping the data consistent. The I/O architecture along with the processor architecture provides the 'tools' to do this.

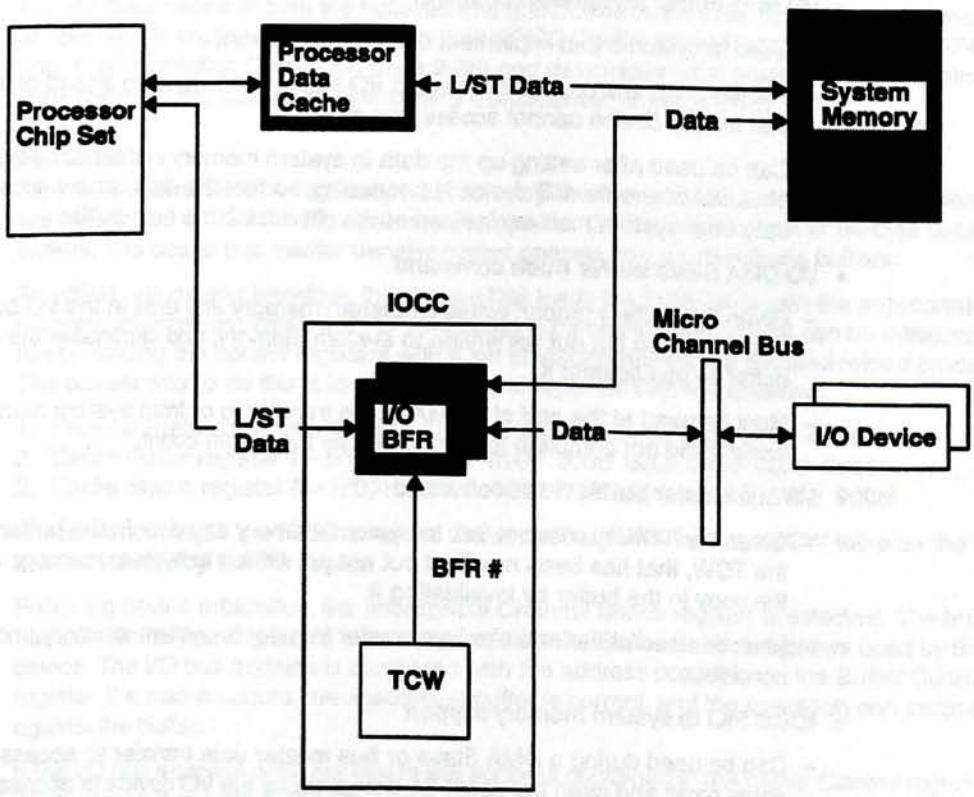


Figure 57. Data Flow in the Programming Model

The following architectural tools assist in providing consistency:

- Processor data cache flush instructions
  - Write data that has been modified in the processor data cache to system memory. Forces subsequent processor accesses to go to system memory, so that you no longer have two different (inconsistent) sets of data in the processor data cache and system memory.
  - Must be executed after setting up I/O data and before a bus master or DMA Slave can read the data. The data is taken from system memory, not the processor data cache.
- Hiding of I/O pages from software processes
  - Can hide the page (4K bytes) of memory so that a software process cannot access it after the data is set up for I/O in the system memory.
  - Works well with large block sizes.

- I/O **buffer invalidate** command
  - Throws away the copy of data in the I/O buffer so that the I/O device cannot access that copy again.
  - Can be used after setting up the data in system memory in the same 64-byte area as the I/O device is accessing, so that the device now accesses the new data from system memory instead of the old data in the I/O buffer.
- I/O **next buffer invalidate** command
  - Used in systems that implement dual buffer support.
  - Throws away the copy of data in the I/O buffer that is reading ahead of the device so that the I/O device cannot access that data.
  - Can be used after setting up the data in system memory in the next 64-byte area ahead of where the I/O device is accessing, so that the device now accesses the new data from system memory instead of the old data in the next buffer.
- I/O DMA Slave **buffer flush** command
  - Given the buffer number, writes to system memory any data in the I/O buffer, that has been modified but not yet written to system memory, and eliminates the copy in the buffer by invalidating it.
  - Must be used at the end of a DMA Slave transfer to or from system memory if the transfer did not complete to termination by the length count.
- I/O bus master **buffer flush** command
  - Given the TCW number, writes to system memory any data in the buffer pointed to by the TCW, that has been modified but not yet written to system memory, and eliminates the copy in the buffer by invalidating it.
  - Must be used at the end of a bus master transfer to system memory under all conditions.
- IOCC PIO to system memory support
  - Can be used during a DMA Slave or bus master data transfer to access data in the same page and even the same 64-byte area as the I/O device is accessing.
  - Guarantees consistency.
  - Works well for small data transfers, but there can be a performance penalty on long data transfers.
- IOCC DMA read-modify-write support
  - Provides support to transfer less than 64 bytes of good data from the I/O buffer to system memory.
  - May be implemented by prefetching the data before the device writes the first byte to the buffer or by postfetching data from memory and merging it with the bytes in the buffer which have been written by the device.
  - Is not atomic with the processor (processor can access the same location in system memory between the IOCC's read and the IOCC's write), so for example, it does not eliminate the need for hiding memory pages.

Bus Master

Bus master transfers refer to data transfers between a bus master I/O device and memory where the bus master device supplies the memory addresses and controls all aspects of the data transfer.

The system I/O architecture supports both buffered and unbuffered bus master transfers. In the buffered mode, I/O buffers are provided as a performance feature and may also include caching of the current TCW table entry in a Buffer Control register. The following sections include descriptions of both the buffered and unbuffered bus master operations. The mode of operation is implementation specific (see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86) and determines what must be done to maintain consistency of the data (see "Maintaining Consistency" on page 2-36).

Buffered Bus Master

Figure 58 shows the bus master operations to system memory. Sequential data transfers are transferred on IOCC buffer boundaries, and the IOCC provides a set of 64-byte data buffers. The actual bus master transfer cycles operate only against these buffers.

To initiate bus master transfers, the system first loads the TCW table with the appropriate mapping information. When the TCW mapping is complete, the channel can be initialized to run by loading the control registers with a set of values starting the demand reload process. The easiest way to do this is to load the control registers with the following:

1. Channel Status register – B'00me 0100 0000 1111 auth auth 0000 0000'
2. Cache Buffer register 4 – B'0000 0000 0000 0000 0000 0000 0000 0000'
3. Cache Status register 8 – B'0010 0000 0000 0000 0000 0000 0000 0000'

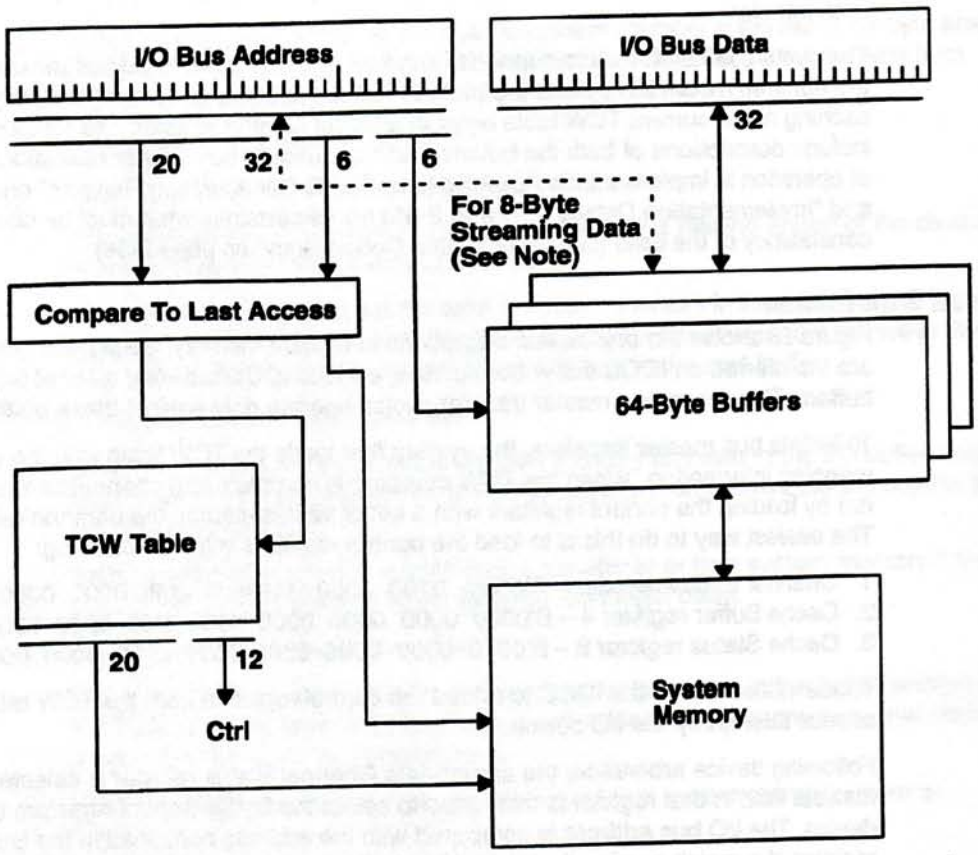
These values cause the IOCC to reload the control registers from the TCW table on the first access attempt by the I/O device.

Following device arbitration, the appropriate Channel Status register is selected. The buffer number field in that register is then used to select the Buffer Control registers used by this device. The I/O bus address is compared with the address contained in the Buffer Control register. If a match occurs, the associated buffer is correct, and the operation can proceed against the buffer.

If the I/O bus address does not match the address contained in the Buffer Control register, a TCW access is required. The I/O bus address is used to select the appropriate TCW, and the buffer number field obtained is used to select the appropriate set of Buffer Control registers. These registers are then tested to see if the I/O address matches. If a match occurs, the contents of the buffer are valid and the operation can proceed. If not, the buffer needs to be loaded.

Prior to loading of the buffer, the current buffer is checked to see if it can be cast out. A bit in the Buffer Control register indicates whether that buffer is dirty. If so, the buffer is written back to system memory prior to access of the new buffer. Following access of a new buffer, the I/O bus address and new TCW are written into the Buffer Control registers.

The IOCC must perform a read-modify-write sequence to guarantee that the buffer space, which has not been written to, does not change the data in system memory when that buffer is written to memory.



Note: Implementation of the Micro Channel 8-byte Streaming Data protocol is optional. (See "Implementation Details" on page 2-86.)

Figure 58. Buffered Bus Master Data Transfer Operation

As shown in Figure 59, each bus master channel is dynamically associated with two 32-bit controlling registers. These registers are also used for DMA slave operations but are defined differently when personalized for bus master data transfer operations.

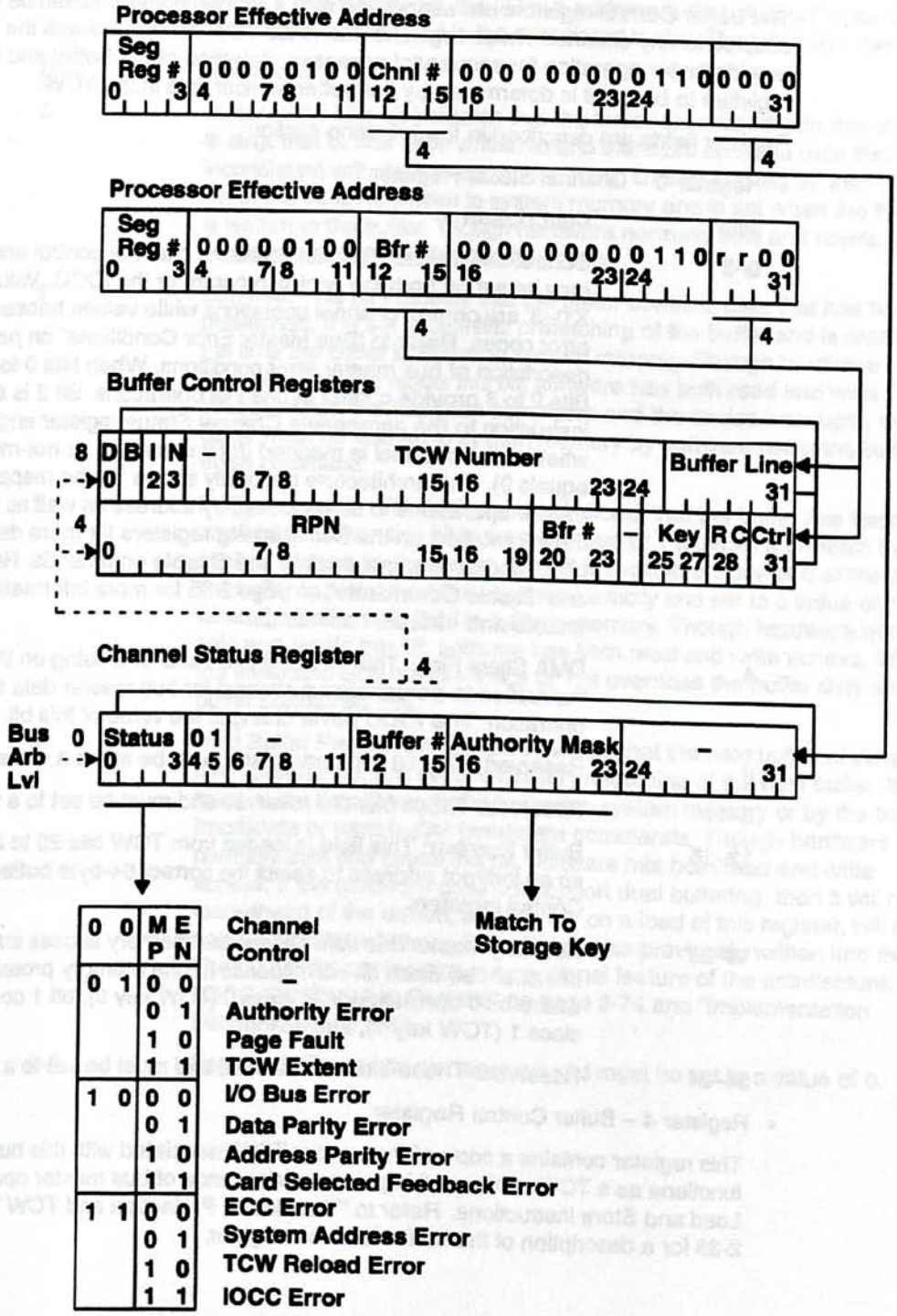


Figure 59. Buffered Bus Master Control Registers

Each of the 16 channels has its own Channel Status register. This register contains channel status, some personalization controls, a buffer pointer, and an 8-bit memory access authority mask.

The Buffer Control registers are associated with a specific buffer and can be dynamically coupled to any channel. These registers cache the TCW associated with the buffer and provide faster operation for sequential accesses. Selection of the Buffer and Buffer Control registers to be used is determined by the buffer number field in the TCW.

Register fields are described in the following section:

- Register 0 – Channel Status Register

Bits	Description
0–3	Control and Status: This field contains channel control and status, and may be set by both the control program or the IOCC. Values between X'0–3' are control channel operations while values between X'04–15' are error codes. Refer to "Bus Master Error Conditions" on page 2-47 for a description of bus master error conditions. When bits 0 to 1 are B'00', Bits 2 to 3 provide control of channel operations. Bit 2 is set by a Store instruction to the appropriate Channel Status register and indicates whether the channel is mapped (Bit 2 equals 1), or not-mapped (Bit 2 equals 0). The architecture optionally allows for the mapping of bus master operations to be controlled by address as well as by channel; see the information on the Bus Mapping registers for more details. Bit 3 is controlled by channel **enable** and **disable** commands. Refer to "enable and disable Commands" on page 2-65 for more information on the **enable** and **disable** commands.
4	DMA Slave Flag: This bit is set to a value of 0 using an I/O Store instruction to personalize a channel for bus master data transfer operation. The IOCC never changes the value of this bit.
5	Reserved: This bit is reserved and must be set to a value of 1.
6–11	Reserved: These bits are reserved and must be set to a value of 0.
12–15	Buffer Number: This field is loaded from TCW bits 20 to 23 and is used as an indirect address to select the correct 64-byte buffer and Buffer Control registers.
16–23	Authority Mask: This field defines the memory access authority granted to this channel. Each bit corresponds to one memory protection class, where bit 0 corresponds to class 0 (TCW key 0), bit 1 corresponds to class 1 (TCW key 1), and so forth.
24–31	Reserved: These bits are reserved and must be set to a value of 0.

- Register 4 – Buffer Control Register

  This register contains a copy of the current TCW associated with this buffer. This register functions as a TCW cache and improves performance of bus master operations and Load and Store instructions. Refer to "Translation, Protection and TCW Table" on page 2-33 for a description of the bit fields in this register.

- Register 8 – Buffer Control Register

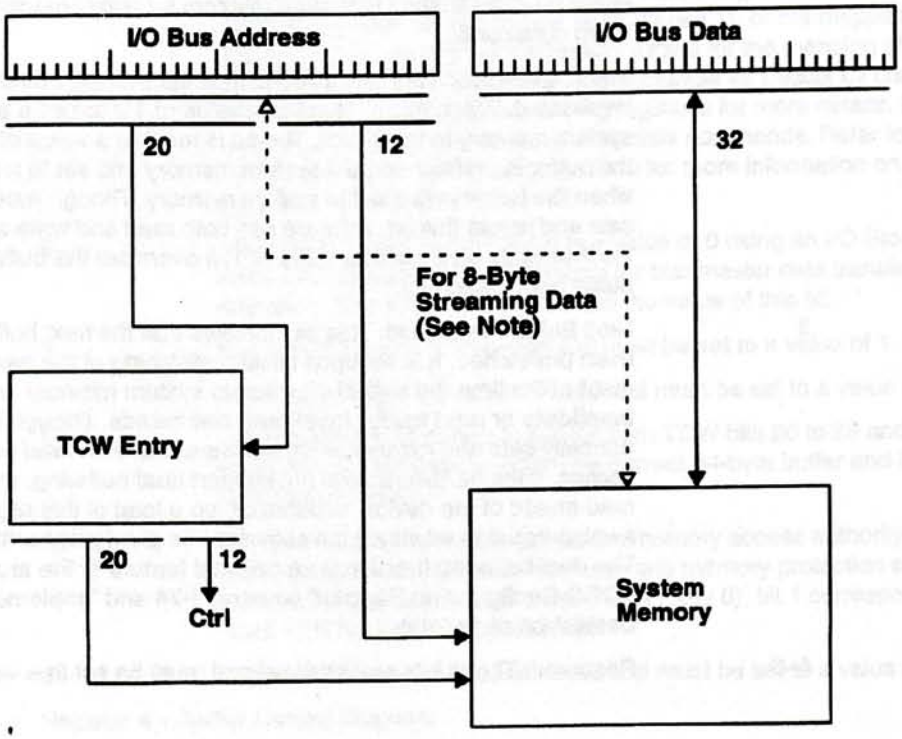
  This register contains a copy of the I/O bus address associated with the TCW register described in the preceding text. Whenever a bus master operation or a Load and Store instruction references a memory object, the I/O bus address is first checked against this register to see if the object is contained in the associated buffer. The bit usage follows:

Bits	Description
0	Buffer Dirty: This bit indicates that the buffer associated with this channel is *dirty*, that is, has been written to and therefore contains data that is inconsistent with data in system memory. This bit is reset by the IOCC when the buffer is written to system memory and is set when the first byte is written to the buffer. Though hardware normally sets and resets this bit, software has both read and write access.
1	Buffered: This bit indicates that the buffer contains data that has been prefetched. It is set upon initial prefetching of the buffer and is reset at the time the buffer is written to system memory. Though hardware normally sets and resets this bit, software has both read and write access. When the operation completes and the device interrupts, the buffer must be flushed to system memory by software using the **buffer flush** command.
2	Buffer Invalidate: This bit is used to indicate that the buffer has been invalidated. When this bit is set to a value of 1 it forces a prefetch from system memory to this buffer. The bit is reset to a value of 0 at the time the buffer is prefetched from system memory and set to a value of 1 when the buffer is flushed to system memory. Though hardware normally sets and resets this bit, software has both read and write access. When the invalidate bit is set to a value of 1, it overrides the buffer dirty and the buffer prefetched bits.
3	Next Buffer Prefetched: This bit indicates that the next buffer of data has been prefetched. It is set upon initial prefetching of the next buffer. It is reset at the time the buffer is flushed to system memory or by the **buffer invalidate** or **next buffer invalidate** commands. Though hardware normally sets and resets this bit, software has both read and write access. If the hardware does not support dual buffering, then it will not read-ahead of the device, and this bit, on a load of this register, will have a value equal to whatever the software has previously written into the bit. The dual buffering function is an optional feature of the architecture; see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86.
4–5	Reserved: These bits are reserved and must be set to a value of 0.

6–25	I/O Bus Address A31 to A12: This field is used by the IOCC to detect when a page changes. It contains a copy of the I/O bus address that caused the last TCW to be fetched. This field is referred to on a cycle-by-cycle basis to determine if the current TCW in register 4 is valid. If a page is changed, that is, address bits A31 to A12 change, the IOCC accesses the TCW table again.
26–31	I/O Bus Address A11 to A6: This field is used by the IOCC to detect when a buffer changes. It contains a copy of the I/O bus address relating to the current 64-byte I/O buffer within the 4 K-byte system page. If a bus master changes buffers within the 4 K-byte system page, that is, address bits A11 to A6 change, the IOCC accesses system memory as appropriate to make a new 64-byte I/O buffer available.

Unbuffered Bus Master

Figure 60 shows the unbuffered bus master operations to system memory. Note that the 64-byte IOCC buffers are not shown as with the buffered mode previously described. The caching of the current TCW table entry is not shown. Figure 60 assumes direct access of the TCW table entry on each I/O access by the bus master.



Note: Implementation of the Micro Channel 8-byte Streaming Data protocol is optional (See "Implementation Details" on page 2-86).

Figure 60. Unbuffered Bus Master Data Transfer Operation

The Bus Master Channel Status register for the unbuffered case is shown in Figure 61. Each of the 16 channels has its own Channel Status register. This register contains status, some personalization controls, and an 8-bit memory access authority mask.

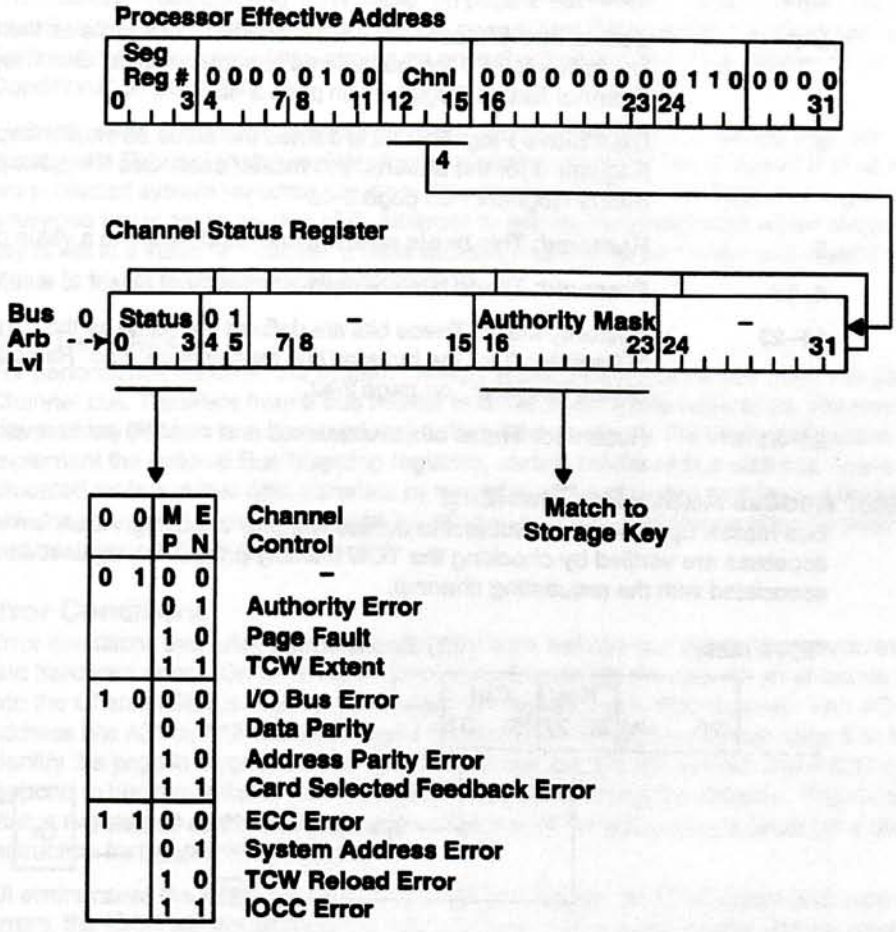


Figure 61. Unbuffered Bus Master Control Registers

Note: The Buffer Control registers shown in Figure 59 on page 2-41 are not supported in this mode. A Load instruction to register 8 returns all zeroes. On a Store instruction to register 8, data is ignored. Register 4 is used for DMA slave operations. A Load or Store instruction to register 4 is treated as described in "DMA Slave" on page 2-49.

Following device arbitration, the appropriate Channel Status register is selected. The I/O bus address is used to select the appropriate TCW. The RPN from the TCW entry and 12 bits from the I/O bus address are used to address system memory.

Register fields are described as follows:

- Register 0 – Channel Status Register

Bits	Description
0–3	Control and Status: These bits are defined the same as the corresponding bits in Register 0 for the buffered bus master case. See "Register 0 – Channel Status Register " on page 2-42.
4	DMA Slave Flag: This bit is defined the same as the corresponding bit in Register 0 for the buffered bus master case. See "Register 0 – Channel Status Register " on page 2-42.
5	Reserved: This bits is reserved and must be set to a value of 1.
6–15	Reserved: These bits are reserved and must be set to a value of 0.
16–23	Authority Mask: These bits are defined the same as the corresponding bits in Register 0 for the buffered bus master case. See "Register 0 – Channel Status Register " on page 2-42.
24–31	Reserved: These bits are reserved and must be set to a value of 0.

Bus Master Access Authority Checking

Bus master operations are subject to access authority checking. As shown in Figure 62, accesses are verified by checking the TCW memory protect key against an authority mask associated with the requesting channel.

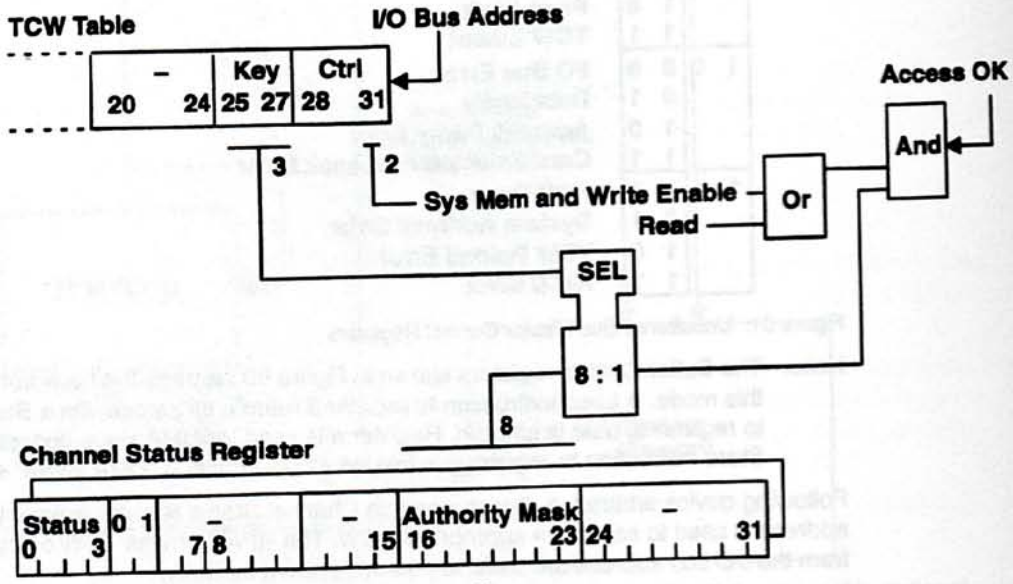


Figure 62. Bus Master Access Authority Checking

Bus master operations are protected on page boundaries. Each page in the bus memory address space has a 3-bit storage protect key associated with that page, that defines the protection class of the page. These keys are kept in the TCW table described in the "Translation, Protection and TCW Table" on page 2-33. An 8-bit mask in each channel specifies the key values (and by inference, pages) that this channel is authorized to access. For information on what action occurs on an authority error, see "Bus Master Error Conditions" on page 2-47.

Authority mask information is considered part of the context and is loaded into the appropriate Channel Status register by the operating system. The Channel Status registers are protected system resources and are only accessible when the Segment register privileged key is set to a value of 0. Attempts to access these registers when the privileged key is set to a value of 1 causes a Data Storage interrupt to be posted and invalid operation status to be set in Channel Status register 15.

Bus to Bus Data Transfers

For performance reasons, the system memory is put on a separate bus from the Micro Channel bus. Transfers from a bus master to an address space have to be directed either to bus memory (for bus to bus operations) or to system memory. For implementations that implement the optional Bus Mapping registers, certain blocks of bus address space can be allocated for bus to bus data transfers by way of the Bus Mapping registers. Alternately, all data transfers from a bus master can be directed to the bus address space by setting bit 2 of that bus master's CSR to a value of 0.

Bus Master Error Conditions

Error conditions that arise in bus master operations include bus errors, programming errors, and hardware errors. On an error, an error code identifying the specific error cause is set into the Channel Status register (bits 0 to 3) corresponding to that channel. The I/O bus address bits A31 to A12 are also logged into the Channel Status register (bits 6 to 25) to identify the page in error. After the error code is set into the status field, the IOCC does not respond to bus requests for this channel, effectively disabling the channel. The Channel Status registers thus capture the channel status until the error code is reset by a Store instruction from the system supervisor.

All errors cause the 'chck' signal to be pulsed. In addition, on TCW extent and address parity errors, the IOCC will not activate the 'sfdbkrtn' line. When a bus master device sees this error condition, it should suspend operations and post an interrupt. For additional information refer to "Exception Reporting and Handling" on page 2-85.

After the error condition, if the bus master device tries to continue accesses with the channel effectively disabled (also, if the bus master tries to make an access and the channel was never enabled), the IOCC activates 'chck' and will not activate 'sfdbkrtn'. If the access is directed to the IOCC, the IOCC does not take or supply data, and continued read accesses by the device after the error results in the IOCC bus drivers being disabled which results in all ones on the I/O data bus.

I/O bus errors such as an address or data parity errors may be caused by hardware malfunctions or transient electrical noise. Refer to "Parity Errors" on page 2-19 and "Channel Check" on page 2-19 for a description of these errors. Error codes are summarized as follows:

Error Code	Description
0 1 0 1	Authority Error: This error code is set if the storage key in the TCW does not match the authority mask in the Channel Status register or an attempt is made to write to a read-only page.
0 1 1 0	Page Fault Error: This error code is set if an attempt is made to access a page with TCW bits 30 and 31 set to B'01'. This can occur in normal operation. Devices attempting to take advantage of this function must present an interrupt after receiving a 'chck' signal on the I/O bus.
0 1 1 1	TCW Extent Error: This error code is set if an attempt is made to access a bus address for which a TCW does not exist.
1 0 0 0	I/O Bus Error: This error code is set if an error on the Micro Channel bus has been detected during a transfer. The types of errors detected here are implementation dependent (see "Implementation Details." on page 2-86.)
1 0 0 1	Data Parity Error: This error code is set if the IOCC detects bad parity when operating as a slave on the bus (when the transfer is from device to system memory).
1 0 1 0	Address Parity: This error code is set if the IOCC detects bad parity on the address bus. This error is detected even when the IOCC is not involved in the transfer (that is, on a bus-to-bus transfer). This is a bus monitoring function of the IOCC.
1 0 1 1	Card Selected Feedback Error: This error code is set if, after a device is addressed it does not respond by driving the 'cd sfbk' line. This is a bus monitoring function of the IOCC.
1 1 0 0	ECC Error: This error code is set if the IOCC received an uncorrectable ECC error response from the system bus during a bus master transfer request to system memory.
1 1 0 1	System Address Error: This error code is set if the IOCC sends data over the system bus and does not receive an address acknowledgement. This can occur if the real page number in the TCW is invalid. Software should make sure that the real page number in the TCW is valid.
1 1 1 0	TCW Reload Error: This error code is set if the IOCC detects a parity or uncorrectable ECC error during a TCW access.
1 1 1 1	IOCC Error: This error code is set if the IOCC detects an internal error (except those dealing with the Channel Status registers or Buffer Control registers) during any bus master channel operation. An error with the Channel Status or Buffer Control registers results in a check stop.

DMA Slave

DMA controller is the name given to a system-supplied resource that mediates data transfers between memory and DMA slaves. The IOCC contains a DMA controller for the I/O bus. Three parties are involved in this type of DMA operation: the DMA slave, the memory, and the DMA controller. This type of DMA operation is often used for the following reasons:

- Cost

  A DMA controller must provide interfaces to both system addresses and data and is highly pin-intensive. The data flow is quite regular and lends itself well to implementation using RAM arrays. Thus, multiple-channel DMA controllers are relatively easy to implement. Since most systems require at least one DMA device, a common practice in low-end systems is to provide a multi-channel DMA controller as a shared resource and amortize its cost across multiple devices.

- Protection

  DMA controllers manage all address, control, and byte count functions associated with data transfer. As such, it is relatively easy for a system to protect its memory from the external environment by using DMA *channels*, and making channel setup a privileged operation.

Using the DMA controller, data can be transferred between a device and bus memory, or between a device and system memory. Data transfers to or from system memory may or may not be buffered. The system I/O architecture supports both buffered and unbuffered DMA slave transfers. In the buffered mode, I/O data buffers are provided as a performance feature for transfers between I/O and system memory, and can also include caching of the current TCW table entry in a Buffer Control register. Data transfers to or from bus memory are never buffered. The following sections include descriptions of both the buffered and unbuffered DMA slave operations. The mode of operation is implementation specific (see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86) and determines what must be done to maintain consistency of the data (see "Maintaining Consistency" on page 2-36).

All memory is partitioned into 4K-byte pages, and the DMA controller is organized to handle physical transfers of this size. The architecture supports two modes of managing each 4K-byte page of memory for DMA slave operations. One mode uses TCWs and the other uses tag elements to handle this management of memory pages. See "DMA Slave Operations Using Tags" on page 2-50 and "DMA Slave Operations Using TCWs" on page 2-57 for a description of these two modes. The choice of using TCWs or tags for the management of the 4K-byte pages is implementation dependent. (See "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86.)

Each DMA slave channel includes a pair of 32-bit registers used to contain the current memory address and control information corresponding to the current page being accessed. The IOCC implements up to 15 DMA channels. Each channel is associated with one of 16 I/O bus arbitration levels. One of these arbitration levels (level 15) must be allocated to the system processor for issuing Load and Store instructions to the I/O bus, reducing the maximum number of useable DMA channels to 15. For implementations using tags, the number of channels implemented must be 15. For implementations using TCWs, the number of useable DMA channels is implementation dependent (see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86).

The DMA Slave Control registers are accessible by way of Load and Store instructions from the system processor, and are located in the IOCC address space. DMA Slave Control registers are a protected system resource and are only accessible when the Segment register privileged key is set to 0. Attempts to access these registers when the privileged key is set to a value of 1 will cause a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

Each channel is personalized to operate with either a bus master or DMA slave. Bit 4 of the Channel Status register (DMA register 0) must be set to a value of 1 when controlling a DMA slave device, and set to 0 when controlling a bus master device.

Note: Software should program unallocated channels as bus master channels.

The system supervisor must first load the DMA slave control registers prior to enabling a channel. Following setup, the channel is enabled using the DMA **enable** command described in the "enable and disable Commands" on page 2-65. The IOCC is then ready to control DMA operations on behalf of a DMA slave device.

The action taken when loading a Channel Status register for DMA slave operation where there are fewer channels than Channel Status registers, with a channel number greater than that indicated in the IOCC Configuration register is implementation-dependent. (See "Implementation Details" on page 2-86.) Software supports assignment of DMA channels to arbitration levels on a first come first serve basis. If a channel is not available, the resource request is rejected. Hardware does not check for the mapping of a DMA channel to more than one arbitration level at a time. This must be controlled by the software.

If the operation completes without error, the IOCC terminates the DMA slave operation and disables the channel. If an error occurs during the DMA slave operation, the IOCC sets a code identifying the error into the Channel Status register status field and terminates the DMA slave operation. No additional DMA slave requests or **enable** commands will be accepted by this channel until the error is cleared by way of a Store instruction. The DMA Slave Control registers are frozen, capturing details on channel status at the time of error. Refer to "DMA Slave Error Conditions" on page 2-62 for details.

To suspend or terminate a DMA operation prior to its normal ending point, it is recommended that a DMA **disable** command be used. This command provides a soft termination of a DMA operation without destroying the current state of the DMA slave control registers. Refer to "enable and disable Commands" on page 2-65 for details on this command.

DMA slave termination is accompanied by the IOCC pulsing the 'tc' signal. Devices are expected to post an interrupt when this occurs, notifying the system that the DMA operation is complete. The system supervisor can then inspect the DMA registers to determine if the operation completed normally.

DMA Slave Operations Using Tags

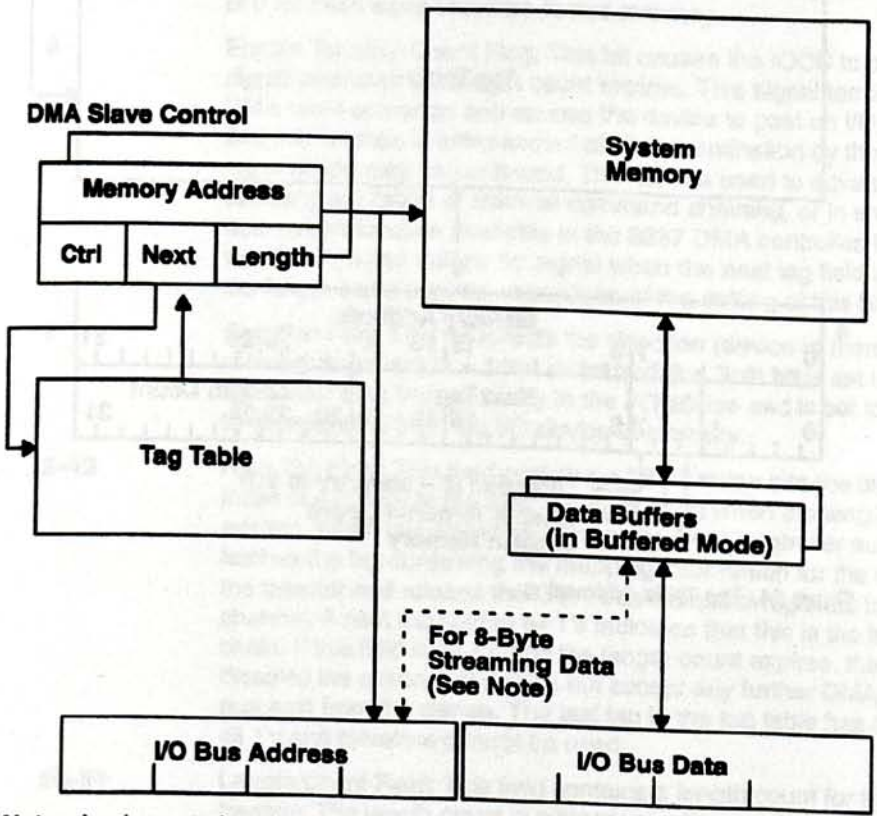
Tags provide support for byte-level scatter and gather DMA slave operations. A DMA slave transfer is described by the DMA Slave Control registers and a list of tag entries. The DMA Slave Control registers describe the initial partial transfer and each of the tags describes another part of the transfer.

DMA Slave Control registers 0 and 4 contain a copy of the tag except for the status field as described in "DMA Slave Error Conditions" on page 2-62 and "enable and disable Commands" on page 2-65.

The tags are organized as a heap in a special memory space called a *tag table*. The tag table includes 4096 entries. During the course of a DMA slave operation, the IOCC will reload the DMA Slave Control registers from the tag table on a demand basis. The DMA Slave registers must be loaded directly using a Store instruction with the initial tag entry.

To allow for management of large logical buffers, the DMA controller allows chaining of tags. Whenever a page boundary is crossed or the length count expires, the DMA controller automatically fetches the tag containing the mapping information for the next page and reloads the DMA Slave Control registers for that channel. Since each tag also includes length count information, this structure provides natural *data chaining* down to the byte level.

Figure 63 shows the DMA slave operations using tag elements. Data may be transferred between a device and system memory or between a device and bus memory. In the buffered mode, the IOCC must provide a 64 byte data buffer (or dual buffer; see "System Structure" on page 2-4) for each channel, and this buffer must be managed by the software. The actual I/O bus DMA cycle operates only against these buffers. In the unbuffered mode, the IOCC must provide some read-modify-write capability so that transfers from the device, that are less than the memory read and write granularity, can be matched to the system memory interface. Data transfers to or from bus memory are not buffered.



Note: Implementation of the Micro Channel 8-byte Streaming Data protocol is optional (See "Implementation Details" on page 2-86).

Figure 63. DMA Slave, Using Tags

The tag table is a protected system resource located in the IOCC address space between addresses X'–0 80 00 00' and X'–0 80 7F FF'. Figure 64 shows this address space. It is only accessible to Load and Store instructions from the system processor when the Segment register privileged key is set to a value of 0. Attempts to access this table when the privileged key is set to a value of 1 causes a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

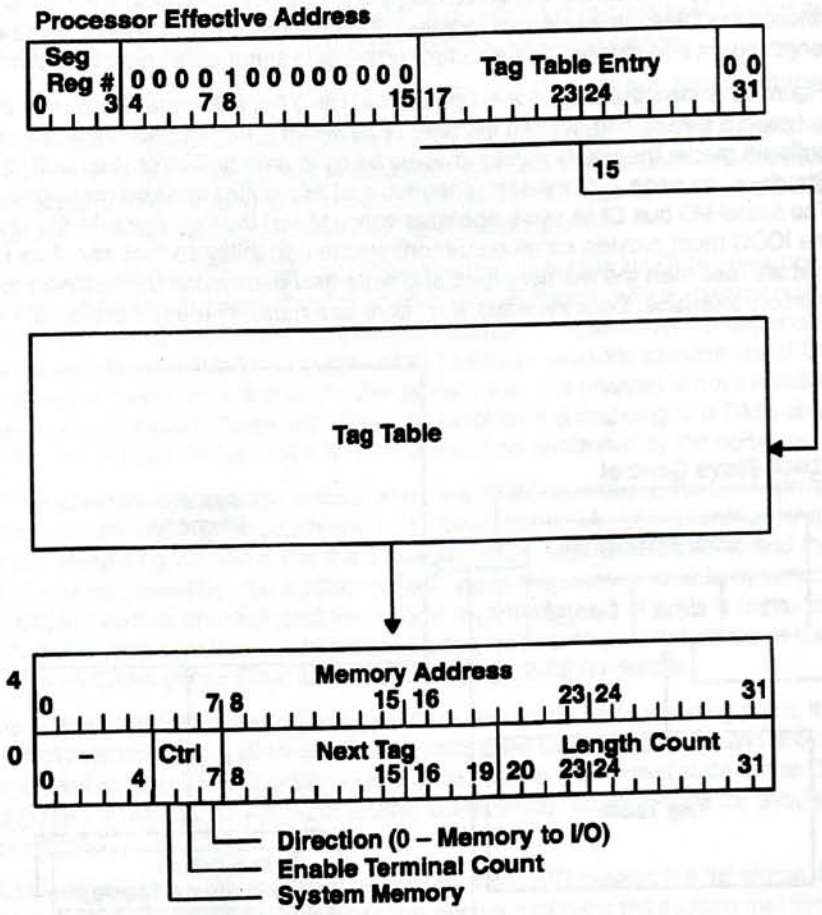


Figure 64. Tag Table Addressing

Each 4K-byte page involved in a DMA slave transfer, except for the first, has at least one 8-byte tag element in the tag table. The first tag is set up in the DMA Slave Control registers. These tags contain relevant information required for the DMA slave operation such as the memory address, length count, and direction. Tags may be chained together to control DMA across multiple memory pages, or to provide a data chaining function. Each tag represents the initial set of values to be loaded into the DMA Slave Control registers every time a page is crossed or the length count of the current transfer expires. Access to the tag table entries is word access only. The bit definition of a tag entry is defined as follows:

- Word 0 of a tag contains control information relating to the current 4K-byte page and includes the following:

Bits	Description
0–4	Reserved: This field is reserved and must be set to a value of 0. The hardware does not update the Channel Status register bits 0 to 3 with these bits.
5	System Memory Flag: This bit selects whether system memory or bus memory is to take part in a DMA slave transaction. This bit is set to a value of 1 for DMA slave transfers to system memory and set to a value of 0 for DMA slave transfers to bus memory.
6	Enable Terminal Count Flag: This bit causes the IOCC to pulse the 'tc' signal whenever the length count expires. This signal terminates the DMA slave operation and causes the device to post an I/O interrupt. Note that this function is independent of DMA termination by the channel, and tag chaining may be continued. This can be used to advantage in assisting emulation of channel command chaining, or in emulating the auto-reload function available in the 8237 DMA controller. Note also that the IOCC always pulses 'tc' signal when the next tag field is X'FFF' and the length count expires, regardless of the setting of this bit.
7	Direction Flag: This bit selects the direction (device to memory or memory to device) of a DMA slave transfer. This bit is set to a value of 0 to transfer data from memory to the I/O device and is set to a value of 1 to transfer data from the I/O device to memory.
8–19	Next Tag Field: This field contains a 12-bit index into the tag table. This index is a pointer to the next tag to be used when the length count expires. When this condition occurs, the DMA controller automatically fetches the tag containing the mapping information for the next piece of the transfer and reloads the DMA Slave Control registers for that channel. A next tag field of all 1's indicates that this is the last tag in a chain. If this field is all 1's and the length count expires, the IOCC disables the channel and does not accept any further DMA slave requests from the device. The last tag in the tag table has an address of all 1's and therefore cannot be used.
20–31	Length Count Field: This field contains a length count for the data transfer. The length count is a binary number one less than the number of bytes to be transferred and cannot be greater than the number of bytes left to the end of the page.

- Word 4 of a tag contains a 32-bit *real* address to either the bus memory space or system memory space.

Figure 65 shows the register definitions when tag control elements are used to manage memory. Bits 28 and 29 *(r)* in the effective address indicate which word is being addressed.

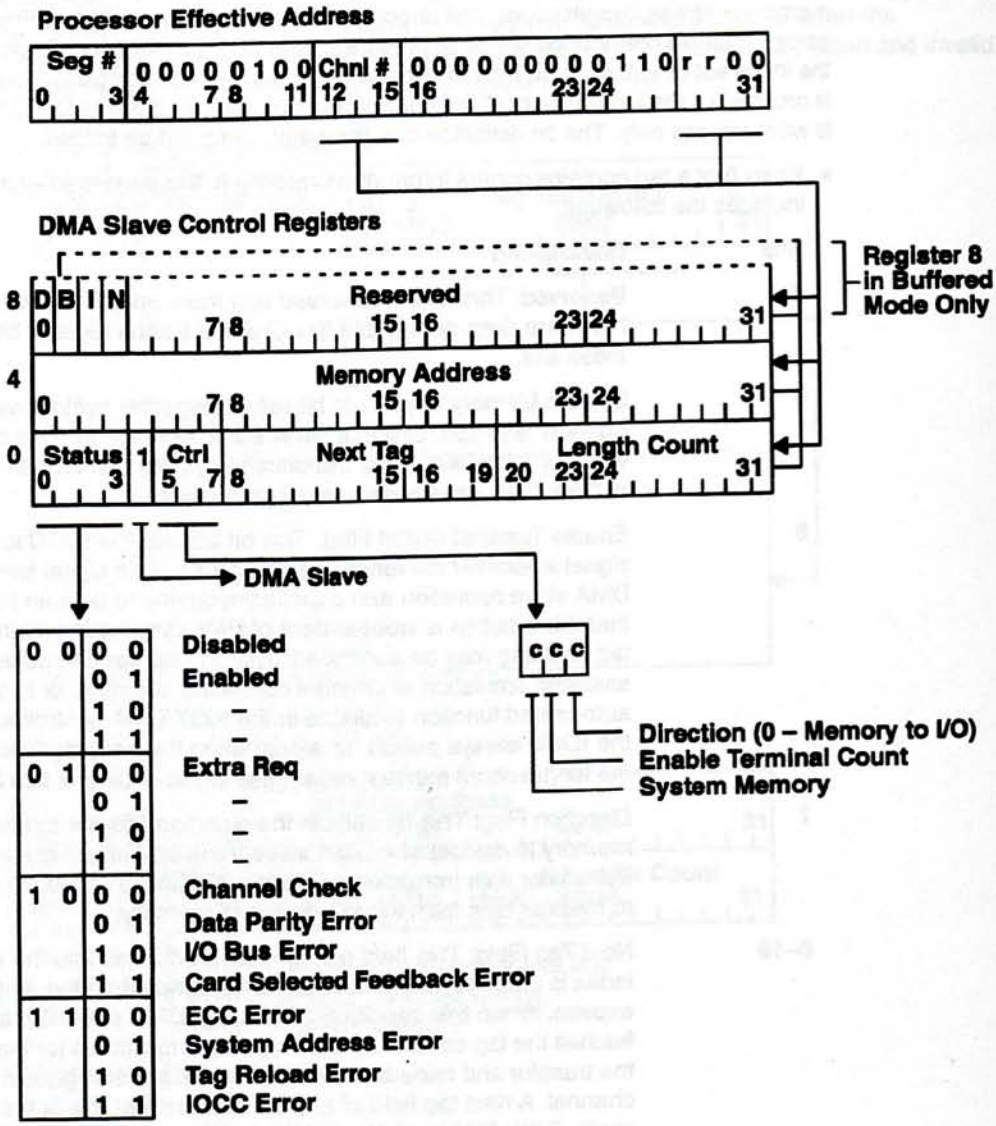


Figure 65. DMA Slave Registers Using Tags

The register fields are described in the following section.

- Register 0 – Channel Status register

  There are 16 Channel Status registers (CSR) each having a one-to one correspondence to one of 16 arbitration levels. The bit assignments for this register are as follows:

Bits	Description
0–3	Control and Status: This 4-bit field contains control information when bits 0 and 1 are B'00'. When bits 2 and 3 are at B'00', the channel associated with this arbitration level is in the disabled state. When bits 2 and 3 are at B'01', the channel is enabled. Bit 3 is set using the channel **enable** command and reset using the channel **disable** command. Code points B'10' and B'11' for bits 2 and 3 are reserved. When bits 0 and 1 are not at B'00', the contents of bits 0 and 3 represents error codes. See "DMA Slave Error Conditions" on page 2-62 for a description of these error codes.
4	DMA Slave Flag: This bit is defined the same as for the tag table word 0 defined on page 2-53.
5	System Memory Flag: This bit is defined the same as for the tag table word 0 defined on page 2-53.
6	Enable T/C Flag: This bit is defined the same as for the tag table word 0 defined on page 2-53.
7	Direction Flag: This bit is defined the same as for the tag table word 0 defined on page 2-53.
8–19	Next Tag Field: This bit is defined the same as for the tag table word 0 defined on page 2-53.
20–31	Length Count Field: This bit is defined the same as for the tag table word 0 defined on page 2-53.

- Register 4 – Memory Address Register

  This register is defined the same as tag table word 4 on page 2-53.

- Register 8 – Buffer Control Register

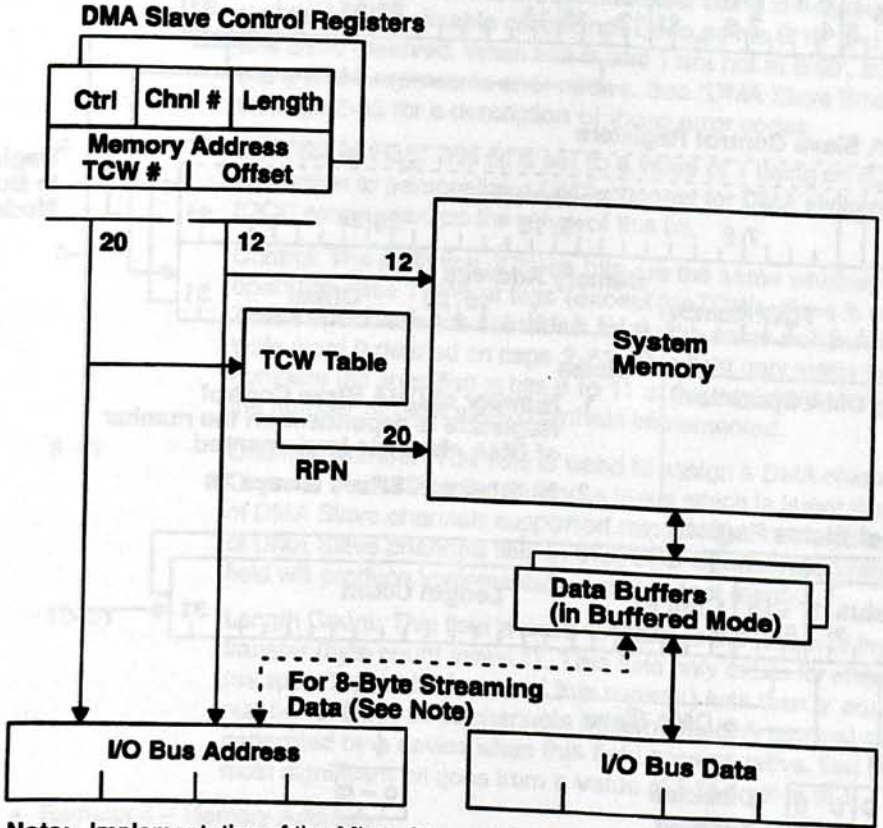
  This register only exists for buffered implementations. The bits assignments are as follows:

Bits	Description
0	Buffer Dirty: This bit is used to indicate that the buffer associated with this channel is *dirty*, that is, has been written to and therefore contains data which is inconsistent with data in system memory.
1	Buffered: This bit indicates that the buffer contains data that was prefetched. It is set upon initial prefetching of the buffer and is reset at the time the buffer is flushed to system memory. Though hardware normally sets and resets this bit, software has both read and write access.
2	Buffer Invalidate: This bit indicates that the buffer was invalidated. When this bit is set to a value of 1 it forces a prefetch from system memory to this buffer. The bit is reset to a value of 0 at the time the buffer is prefetched from system memory and set to a value of 1 when the buffer is flushed to system memory. Though hardware normally sets and resets this bit, software has both read and write access.
3	Next Buffer Prefetched: This bit indicates that the next buffer of data has been prefetched. It is set upon initial prefetching of the next buffer. It is reset at the time the buffer is flushed to system memory or by the **buffer invalidate** or **next buffer invalidate** commands. Though hardware normally sets and resets this bit, software has both read and write access. If the hardware does not support dual buffering, then it will not read-ahead of the device, and this bit will always be returned as a value of 0 on a load of this register. The dual buffering function is an optional feature of the architecture; see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86.
4–31	Reserved: These bits are reserved and must be set to a value of 0.

DMA Slave Operations Using TCWs

TCWs provide support for page level scatter and gather DMA slave operations. The DMA Slave Control register is initialized with the first page TCW; the rest of the TCWs involved in the transfer are sequential. Figure 66 on page 2-57 shows the DMA slave operations using TCWs. Notice that the memory address consists of a TCW number and an offset (unlike the tag which contains a *real* address to system memory).

When TCW entries are used for DMA slave operations, bits 20 to 31 of the TCW entry are not used and software must set these to a value of 0. See "Translation, Protection and TCW Table" on page 2-33 for a description of the TCW table.



Note: Implementation of the Micro Channel 8-byte Streaming Data protocol is optional (See "Implementation Details" on page 2-86).

Figure 66. DMA Slave, Using TCWs

Figure 67 on page 2-58 shows the register definitions when TCWs are used to control DMA slave operation.

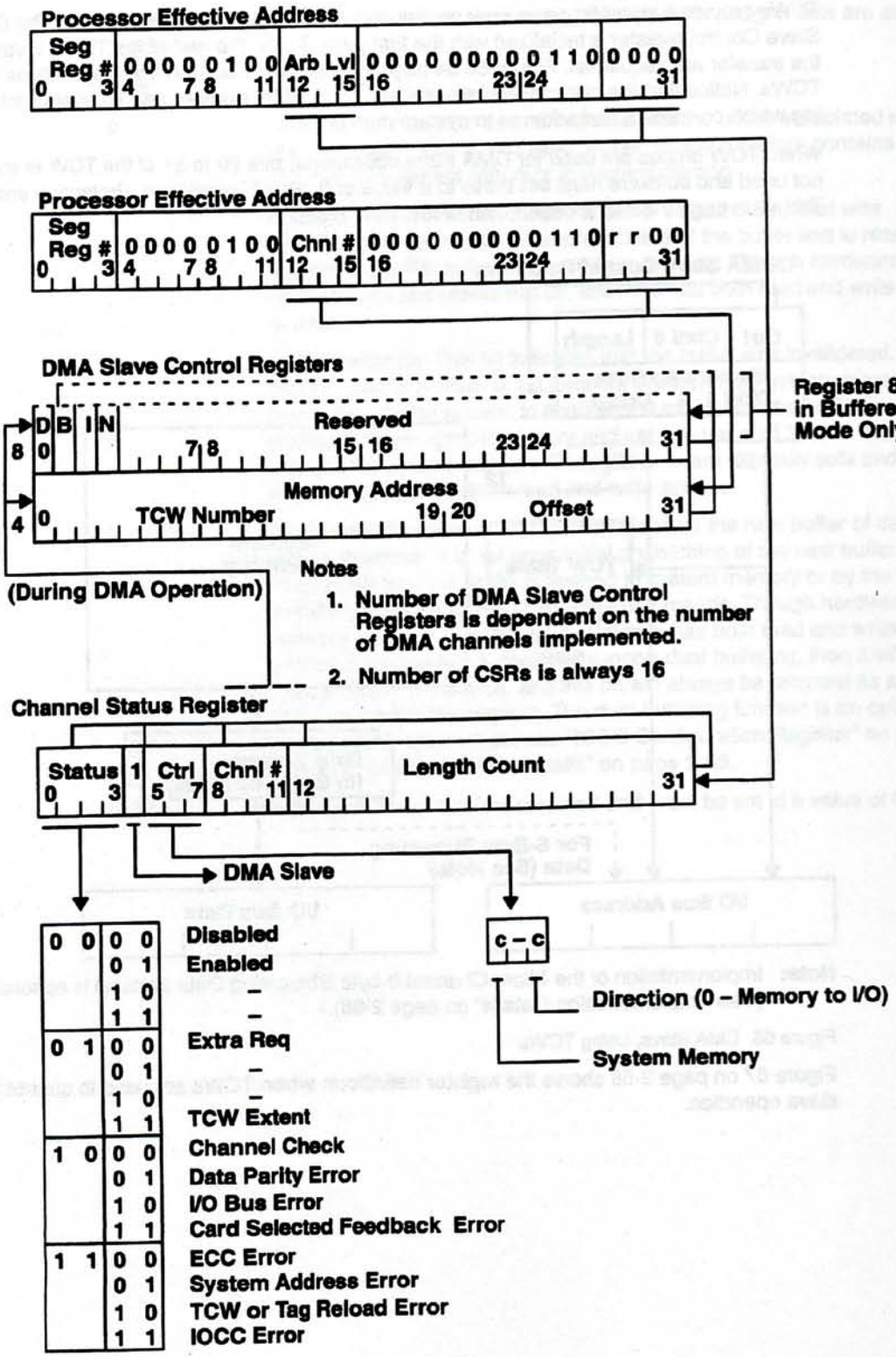


Figure 67. DMA Slave Registers Using TCWs

The register fields are described in the following section.

- Register 0 – Channel Status register

  There are 16 Channel Status registers (CSR) each having a one to one correspondence to one of 16 arbitration levels. The bit assignments for this register are as follows.

Bits	Description
0–3	Control and Status: This 4-bit field contains control information when bits 0 and 1 are B'00'. When bits 2 and 3 are at B'00', the channel associated with this arbitration level is in the disabled state. When bits 2 and 3 are at B'01', the channel is enabled. Bit 3 is set using the channel **enable** and reset using the **disable** command. Code points B'10' and B'11' for bits 2 and 3 are reserved. When bits 0 and 1 are not at B'00', the contents of bits 0 and 3 represents error codes. See "DMA Slave Error Conditions" on page 2-62 for a description of these error codes.
4	DMA Slave Flag: This bit is set to a value of 1 using an I/O Store instruction to personalize a DMA channel for DMA slave operation. The IOCC never changes the value of this bit.
5–7	Control: The definition of these bits are the same whether the DMA slave operation uses TCWs or tags (except for TCWs, there is no T/C enable). These operations are described under the same numbered bits of tag table word 0 defined on page 2-53. This field only exists for channel numbers (as specified in bits 8 to 11 of this register) less than or equal to the number of DMA slave channels implemented.
8–11	Channel Number: This field is used to assign a DMA channel to a specific Channel Status register. Storing a value which is larger than the number of DMA Slave channels supported minus 1 (as indicated by the number of DMA Slave channels field in the IOCC Configuration register) to this field will produce implementation-dependent results.
12–31	Length Count: This field is used to indicate the length of the DMA slave transfer (byte count minus 1). This field only exists for channel numbers (as specified in bits 8 to 11 of this register) less than or equal to the number of DMA slave channels implemented. A terminal count is generated by a device when this field goes negative, that is, when the most significant bit goes from a value of 0 to a value of 1.

- Register 4 – Memory Address

  This register contains the memory address for the DMA slave operation. The number of registers available of this type is implementation dependent (see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86). However, the number available must equal the number of DMA channels implemented. These registers are dynamically associated to the arbitration level based on the channel number assigned in the Channel Status register (CSR). Software must insure that the same channel number is never assigned to more than one CSR (arbitration level) at any given time.

If the transfer is to or from bus memory (Channel Status register bit 5 equal to 0) this register is applied as a 32-bit address directly to the I/O address bus. If the transfer is to or from the system memory, this register is defined as follows:

Bits	Description
0–19	TCW Number: The TCW number in the memory address provides an index into the TCW table where the RPN is obtained if the channel is mapped to system memory. When mapped to system memory, the address used to address system memory consists of the RPN from the TCW concatenated with the offset.
20–31	Offset: These bits are the lower 12 bits of the memory address.

The DMA address is incremented by the size of the transfer, and the length count is decremented by the same amount. Each time the TCW number is incremented in register 4, the next sequential TCW entry is obtained. Note that if software tries to access register 4 with a channel number greater than the number of channels supported (as indicated in the IOCC Configuration register), the results are implementation-dependent (see "Implementation Details" on page 2-86). Also note that only one DMA channel can be assigned per arbitration level.

- Register 8 – Buffer Control Register

  This register only exists for buffered implementations. The bit assignments are described in "Register 8 – Buffer Control Register" on page 2-56.

DMA Slave Bus Protocols

Conventional bus protocols are used in DMA operations and are documented in "Basic Transfer Cycle" on page 2-17.

I/O devices request DMA service on a demand basis by arbitrating for the bus using the 'preempt' line. This causes the 'grant' line to be deactivated, causing an arbitration cycle. When the 'grant' line is reactivated, the IOCC inspects the Control register associated with the bus requester to determine if any DMA service is required. If it is, the IOCC performs a DMA slave sequence on behalf of the requester.

Typical requests are for one or two bytes. On occasion, multiple requests from different devices are received at the same time. When this occurs, service is sequential with the highest priority requester serviced first.

When service is granted to a device, data is transferred between the device and memory. The sequence to be used depends on whether the memory is bus or system memory. The number of bytes transferred is generally equal to the data width of the device. The DMA address is incremented by the size of the transfer and the length count is decremented by the same amount.

If the specified DMA address does not have the same boundary as the I/O device data width, the operation proceeds using a Partial Transfer Protocol as described in "Partial Transfer Cycles" on page 2-18. For example, a DMA transfer involving a 2-byte I/O device and a buffer starting on an odd address results in two 1-byte DMA sequences being performed. This retains the functional integrity of the operation, but requires additional time to complete the operation. As a result, it is suggested that buffers in system memory be located on address boundaries matching the physical width of the I/O device.

DMA Slave Transfers to Bus Memory

DMA slave transfers between a device and bus memory consist of two bus cycles: one to read the data from the source and one to write the data to the target. An input operation consists of an I/O device read cycle followed by a bus memory write cycle. An output operation is reversed.

There is no buffering on transfers to or from bus memory.

DMA Slave Transfers to System Memory

DMA slave transfers between a device and system memory have only one apparent bus cycle: an I/O device read or write cycle. These transfers are described as follows:

- Buffered

  The memory operation is directed to the IOCC buffer and does not appear as a bus cycle. The buffer operation is overlapped with the I/O cycle, and a sequence of DMA cycles to system memory appears on the bus as a sequence of I/O read or write operations. As a result, the average instantaneous performance of DMA slave transfer to system memory may be much better than to bus memory..

  Whenever the address crosses an IOCC buffer boundary or the length count expires, the IOCC transfers the data between the buffer and system memory. This operation may increase the worst case bus latency (depending on the IOCC implementation), decreasing effective DMA performance.

  No restriction is placed on having DMA addresses begin or end on IOCC buffer boundaries. The DMA controller performs read-modify-write sequences to system memory as required. As this potentially occurs only on the first and last buffers to be transferred, addressing has little effect on performance.

  When performing DMA slave transfers to system memory, and the first address does not start on a 64-byte boundary or the remaining count is less than 64, the DMA controller automatically performs either a buffer prefetch before storing the DMA data into the buffer or does some sort of read-modify-write before storing the data to system memory (depending on the implementation). If a **buffer flush** command is issued before the length count expires and the buffer cache contains less than 64-bytes (the memory address is not B'xx..xx000000'), the remainder of the buffer transfer to system memory may consist of zeros (implementation dependent). See "Buffer Flush Commands" on page 2-66 for additional details.

- Unbuffered

  DMA slave transfers between a device and system memory have only one apparent bus cycle: an I/O device read or write. The memory operation is directed to the IOCC, is overlapped with the I/O cycle, and therefore does not appear as a bus cycle. As a result, the average instantaneous performance of DMA slave transfers to system memory may be twice that of bus memory.

Special Sequences

Special mechanisms are provided to improve the relative data transfer efficiency of highly buffered devices.

The Micro Channel supports preemptive burst operations to take advantage of low average I/O bus loading. A device starts this mode by activating the 'burst' line prior to the end of the DMA slave cycle. No arbitration cycle occurs, and the DMA controller concatenates successive DMA sequences until the 'burst' line is deactivated. Micro Channel arbitration rules require preemptive burst devices to deactivate the 'burst' line request if any other device requires bus service.

The DMA controller also supports a special transfer mode called streaming data transfer. This mode is a single-address, multiple-data protocol, and is described in "Streaming Data" on page 2-17.

DMA Slave Error Conditions

Error conditions that arise in DMA operations include bus errors, programming errors, and hardware errors. The specific cause of the error is coded and set into the status field (bits 0 to 3) in the Channel Status register. The 'tc' signal is then pulsed, which should cause the I/O device to suspend DMA operations and post an interrupt. If it does not, but continues to request DMA service, the IOCC services the DMA requests with dummy cycles, pulsing the 'tc' signal on every cycle. Error codes are summarized as follows:

Error Codes	Description
0 1 0 0	Extra Request Error: This error code is set if a DMA slave request is received by a DMA channel when the channel is disabled. Receipt of an unsolicited DMA request is an error unique to a DMA slave. This error is generally caused by I/O device malfunctions and the IOCC pulses the 'tc' signal in an attempt to shut off the DMA slave. This error can also occur with incorrect programming of the channel.
0 1 1 1	TCW Extent Error: This error code is set if a DMA slave request is received and the DMA slave control register 4 contains a TCW number for which there does not exist a corresponding TCW.
1 0 0 0	Channel Check Error: This error code is set if the device responds with a channel check indication during a DMA slave operation. As an example, a device might respond with a 'chck' signal for a Write operation to that device where there is bad parity on the data, or for other device-detected errors during an operation to that device. This error will not be reported if a card selected feedback error is reported (a card selected feedback error takes precedence over a channel check error).
1 0 0 1	Data Parity: This error code is set if the IOCC detects bad parity on the data bus when the IOCC is reading data. (See "Exception Reporting and Handling" on page 2-85 for details.)
1 0 1 0	I/O Bus Error: This error code is set if an error on the Micro Channel bus has been detected during a transfer. The types of errors detected here are implementation dependent see "Implementation Details" on page 2-86).

Error Codes	Description
1 0 1 1	Card Selected Feedback Error: This error code is set if, after a device is addressed, it does not respond by driving the 'cd sfbk' line. Conditions that could cause this to occur are: if the device is not present; is not seated in the card slot properly; is not enabled or detects bad address parity and does not respond to that address. This error code takes precedence over a channel check error.
1 1 0 0	ECC Error: This error code is set if the IOCC receives an uncorrectable ECC error response from the system I/O bus during a DMA slave request to system memory.
1 1 0 1	System Address Error: This error code is set if the IOCC sends data over the system I/O bus and does not receive an address acknowledgement. This can occur if the real page number in the address is invalid.
1 1 1 0	TCW or Tag Reload Error: This error code is set if the IOCC detects a parity or uncorrectable ECC error during a TCW or Tag table access.
1 1 1 1	IOCC Error: This error code is set if the IOCC detects an internal error during any DMA slave operation. If the IOCC error is on access to the DMA Slave registers; this error will not occur and the machine will check stop instead.

IOCC Commands

IOCC commands are used to change the state of the IOCC or control special bus actions. They take the form of Load and Store instructions to special (effective) addresses, where the addresses specify the actions to be taken. In most cases, the Load or Store instruction can be either a string or nonstring operation. The IOCC include supports the following commands:

- **time delay**
- **end of interrupt**
- **enable** and **disable**
- **buffer flush**
- **buffer invalidate**
- **next buffer invalidate.**

User applications can only issue the **time delay** command, and then only if they have Segment register authority to access the I/O bus. All the other commands are protected and must have the Segment register privileged key set to a value of 0 (bit 1) and the IOCC select bit set to a value of 1 (bit 24). IOCC commands are not placed on the I/O bus.

All IOCC commands are 4-byte operations except the **time delay** command,which can be 1, 2, or 4 bytes.

Time Delay Command

A number of Micro Channel devices have strict rules regarding minimum periodicity of programmed I/O commands. Using program path lengths for timing is not a good programming practice, since program performance varies widely by processor type and (current) operating environment. To assist in programming devices with real-time dependencies, the IOCC supports a special **time delay** command that can guarantee separation of bus I/O commands.

The **time delay** command is coded as a 1-, 2-, or 4-byte Load or Store instruction and is shown in Figure 68 on page 2-64. It is normally inserted between successive Load and Store instructions to devices with time sensitivities and enforces minimum time spacing between the I/O bus cycles. This command is similar to the **time delay** command in the RT system but allows additional time delay increments. The command provides delay increments ranging from 1 to 8 microseconds and is specified using the effective address and the logical (byte) length. If a Load instruction is used to call the time delay function, the data returned is indeterminate. If a Store instruction is used, the data is ignored.

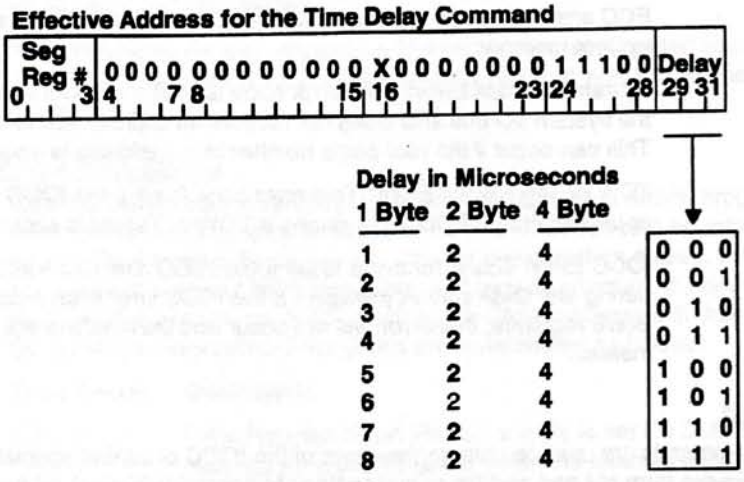


Figure 68. Time Delay Command

The **time delay** command is issued by any user application having Segment register authority to access the I/O bus. Command execution is overlapped with succeeding processor instructions as long as they do not attempt to access any I/O space. If, however, another I/O Load or Store instruction is issued to the I/O space before the time delay has expired, that command is synchronously halted until the pending delay is completed. This command affects only programmed I/O and has no effect on DMA or other I/O operations run by hardware.

The **time delay** command is issued with the I bit in the I/O Segment register equal to 1 or 0. The **time delay** command can be a string operation, but the length must be 1, 2, or 4 bytes. Implementation accuracy of the **time delay** command is to –0 and +1 microseconds (for example, a 1 microsecond delay is greater than or equal to 1 microsecond but less than 2 microseconds).

End of Interrupt Command

Following presentation of an I/O interrupt to the system External Interrupt Source (EIS) register, the IOCC automatically masks off that interrupt so the presentation is only made once. An **end of interrupt** command reenables this mask, causing any active interrupts to be presented (or re-presented) to the system EIS register. On a Store instruction, the data is

ignored. On a Load instruction, the data is indeterminate. This command, shown in Figure 69, should be issued following the interrupt service.

Effective Address for the End of Interrupt Command

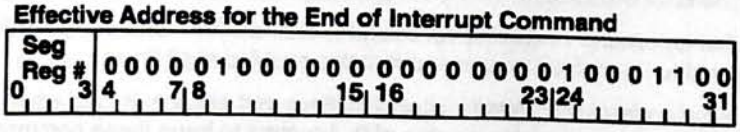


Figure 69. End of Interrupt Command

This command is privileged and is only accessible when the Segment register privileged bit is set to a value of 0. Attempts to run this command when the Segment register privileged bit is set to a value of 1 causes a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

Enable and Disable Commands

The **enable** and **disable** commands allow system initiation and suspension of DMA slave and bus master operations for devices attached to the Micro Channel. Each command is directed to a specific channel as specified by the channel field in the effective address. The command formats are shown in Figure 70. Bits 12 to 15 of the effective address specify the channel to be started or stopped.

Effective Address for the Enable (Load) and Disable (Store) Commands

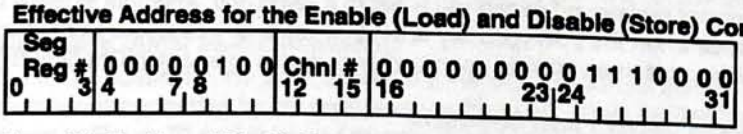


Figure 70. Enable and Disable Commands (Load equals enable and Store equals disable).

The **enable** command initializes a channel to accept requests by changing the channel status in the Channel Status register from the disabled (B'00X0') state to the enabled (B'00X1') state. This command is coded as a Load instruction and returns the original contents of the selected Channel Status register to the target processor register. The channel status field must initially be B'00X0' for this command to update the channel status to the enabled state. This command always returns a status consisting of the full contents of the associated Channel Status register. The status field is the only field changed by this command.

The **disable** command disables operation for a particular channel by changing the channel status from the enabled state (B'00X1') to the disabled (B'00X0') state and is coded as a Store instruction (data is ignored). It does not disrupt any other data in the channel registers, allowing restart of the operation if the device is designed accordingly. The channel status field must initially be B'00X1' for this command to be run. If it is not B'00X1', a no operation (NOP) instruction occurs when this command is issued.

The X in the preceding paragraphs does not indicate a *do not care* state, but indicates that the **enable** and **disable** commands do not change the current state of the status bit 2 (mapped or not-mapped).

A request from a DMA slave when the channel is disabled is considered to be an error and sets an extra request error code in the Channel Status register associated with that device. The 'tc' signal on the Micro Channel bus is pulsed in an attempt to shut off the device.

If a bus master makes a request to a disabled bus master channel, the IOCC does not activate the 'sfdbkrtn' signal and synchronously activates the 'chck' signal, but does not update the error status.

Notice that an **enable** or **disable** command to channel X'F' results in an NOP. Channel X'F' is dedicated to the default master and remains enabled at all times.

These commands are protected system functions and are only issued when the Segment register privileged key is set to a value of 0. Attempts to issue these commands when the privileged key is set to a value of 1 causes a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

Buffer Flush Commands

The **buffer flush** commands are provided for implementations that support IOCC buffers. If the buffers are supported, the IOCC buffers must be flushed to system memory at the end of a transfer. The **buffer flush** commands provide the flush and invalidate functions. Using these commands will result in a NOP (data ignored on a Store instruction, indeterminate on a Load instruction) if the buffers are not supported. For more information on why and when these commands might be necessary, see "Maintaining Consistency" on page 2-36 and "Implementation Details" on page 2-86.

The **buffer flush** commands are protected system functions and can only be issued when the Segment register privileged key is set to a value of 0. Attempts to issue these commands when the privileged key is set to a value of 1 causes a Data Storage interrupt (DSI) to be posted and invalid operation error status to be set in Channel Status register 15.

Bus Master Buffer Flush Command

IOCC buffers for bus master transfers are managed similar to the processor data cache, and a flush operation is performed by address. To improve performance, the **buffer flush** command is defined so the buffer flush can be performed simultaneously with normal TCW maintenance. The command utilizes a bit in the effective address to optionally flush the buffer while accessing a TCW table entry. Figure 71 shows the effective address format. The buffer associated with the TCW is conditionally transferred to system memory if the buffer data has been changed (only flushed if *dirty* and valid). The IOCC remains busy until the buffer transfer is completed and does not accept any new commands. Independent of whether the transfer takes place or not, the buffer is invalidated by setting Buffer Control register 8 to 0 including the D, B, and N bits, the TCW number and the offset, but not including the invalidate bit (I) which gets set to a value of 1. This causes any subsequent accesses to this buffer to have to access again the TCWs and system memory. If on, the Dirty bit is turned off, so any subsequent **buffer flush** commands will not cause a buffer transfer.

Effective Address for the Bus Master Buffer Flush Command

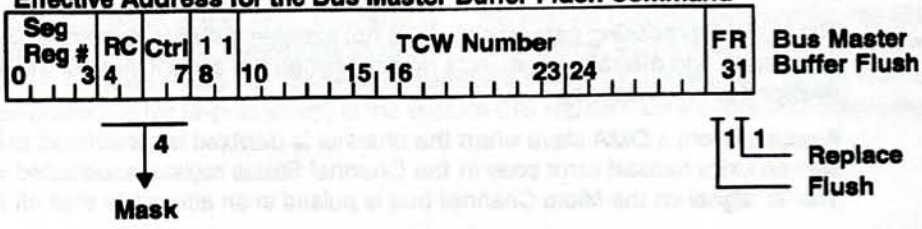


Figure 71. Bus Master Buffer Flush

Bit 30 of the effective address causes any buffers associated with this memory page to be flushed, while bit 31 causes the 4-bit mask value to replace the reference, change, and control bits in the TCW. The following list shows what happens for the various combinations of the Flush and Replace bits:

- Flush equals 0, Replace equals 0.

  This is just a Load or Store instruction to the TCW table.

- Flush equals 0, Replace equals 1.

  The TCW is updated based on the R, C, and CTL bits in the mask field. On a Load instruction, return the old value of the TCW. On a Store instruction, data is ignored.

- Flush equals 1, Replace equals 0.

  On a Load instruction, return the old value of the TCW. If operating in buffered mode, flush the buffer, update the Buffer Control registers, and on a Store instruction, ignore the data. In unbuffered mode, the Store instruction is a NOP.

- Flush equals 1, Replace equals 1.

  On a Load instruction, return the old value of the TCW. On a Store instruction, data is ignored. If operating in buffered mode, flush the buffer, update the Buffer Control registers. The TCW is updated based on the R, C, and CTL bits in the mask field.

DMA Slave Buffer Flush Command

The IOCC buffers for the DMA slaves are managed as simple buffers, and the flush operation is performed by channel number. The DMA Slave **buffer flush** command is shown in Figure 72 and is issued by way of an I/O Store instruction. Bits 12 to 15 of the effective address specifies the buffer that the command is directed to.

Effective Address for the DMA Slave Buffer Flush Command

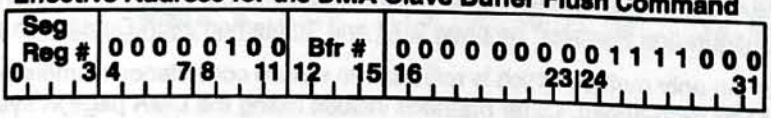


Figure 72. DMA Slave Buffer Flush

The DMA Slave **buffer flush** command conditionally causes the buffer associated with the specified DMA channel to be transferred to system memory if the buffer data has been changed, that is, the Dirty bit is on. The IOCC remains busy until the buffer transfer is completed and does not accept any new commands. Independent of whether the transfer takes place or not, the buffer is invalidated by setting Buffer Control register 8 D, B, and N bits to a value of 0, and the invalidate bit (I) equal to a value of 1.

On a Store instruction, the data is ignored. A Load instruction causes a Data Storage Interrupt. In the unbuffered mode, a Store instruction is a NOP and a Load instruction returns indeterminate data.

Buffer Invalidate Command

Figure 73 shows the effective address format for this command.

Effective Address for Buffer Invalidate Command

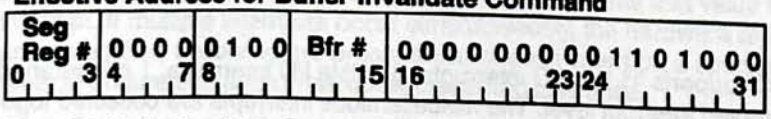


Figure 73. Buffer Invalidate Command

The **buffer invalidate** command assists in the management of DMA slave and bus master operations. This command forces the hardware to reload the buffer on the next DMA slave operation or bus master operation. On bus master operations, the Buffer Control register 4 is also reloaded. A Load instruction returns the state of the bits, but does not invalidate the buffer. On a Store instruction, the data must be X'20000000'. (This is just a store to buffer control register 8 with the buffer invalidate bit turned on.)

If operating in the unbuffered mode, this Store instruction is a NOP, and a Load instruction returns zeroes.

This command is privileged and is only accessible when the Segment register privileged bit is set to a value of 0. Attempts to use this command when the Segment register privileged bit is set to a value of 1 causes a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

Next Buffer Invalidate Command

Figure 74 shows the effective address format for this command.

Effective Address for Next Buffer Invalidate Command

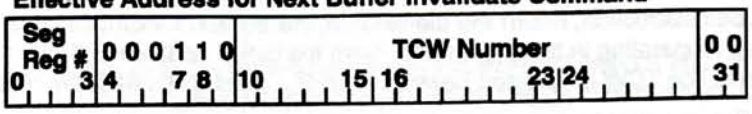


Figure 74. Next Buffer Invalidate Command

The **next buffer invalidate** command is provided to assist in the management of bus master operations. This command forces the hardware to throw away any buffers of data that were read-ahead of where the bus master device is currently reading (the next bit is turned off). This is useful to ensure consistency between the IOCC next buffer and data that may have been modified in system memory. Note that the hardware will read-ahead of the device only if the implementation supports the dual buffering option of the architecture (see "IOCC Configuration Register" on page 2-74 and "Implementation Details" on page 2-86).

This is not the only method which is available to ensure consistency in implementations which support read-ahead. Other methods include hiding the DMA page in system memory from the processor during the transfer to the device and the use of PIO to system memory.

This command must be issued with a full word Store instruction. The data must be a value of 0. A Load instruction causes a Data Storage Interrupt.

If operating in the unbuffered mode or if operating in buffered mode but the dual buffer option is not supported, this command causes a Data Storage Interrupt.

If operating in the buffered mode with dual buffer support, this command is guaranteed not to return an error to the processor (with the exception of a privileged error). Should an error occur, this command invalidates the next buffers for all 16 buffers instead of returning an error indication.

This command is privileged and is only accessible when the Segment register privileged bit is set to a value of 0. Attempts to use this command when the Segment register privileged bit is set to a value of 1 causes a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

I/O Interrupts

The IOCC supports 11 bus I/O interrupts, 3 native I/O interrupts, 1 miscellaneous interrupt, and 1 reserved interrupt level. The miscellaneous interrupts are collected together and are presented as one logical level. This results in a total of 16 IOCC interrupt levels.

The architecture supports both a direct and a coded mapping of the I/O interrupt requests (IRQs) to the External Interrupt Summary (EIS) register. The specific approach supported is implementation dependent (see "Implementation Details" on page 2-86). When the direct mapping approach is supported, the mapping is a direct one for one map (Interrupt level 0 maps directly to EIS bit 0, level 1 maps directly to EIS bit 1 and so on).

The following information describes the coded mapping approach in detail including a description of an Interrupt Vector table used in the mapping.

When the coded mapping is supported, the 16 interrupt levels are coded and are mappable to any EIS bit between 0 and 63. Figure 75 shows the interrupt mechanism.

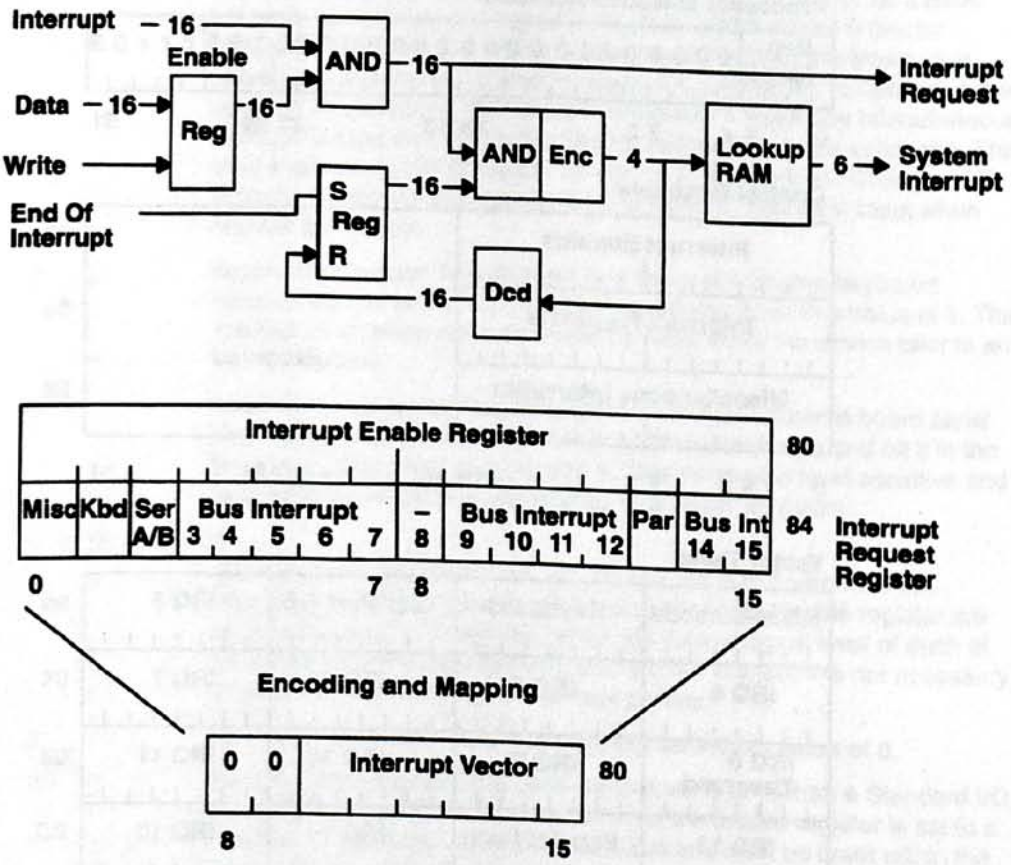


Figure 75. Interrupt Mechanism

Interrupts are presented to the system with a special sequence, setting a bit in the system EIS register corresponding to the vector code presented. Refer to Chapter 1, "System Processors," in this manual, for additional details.

The presentation cycle begins when an interrupt occurs. If the interrupt is enabled, its corresponding bit in the interrupt request field is set to a value of 1. An IOCC sequence then codes the interrupt, looks up a vector value, and presents that value to the system as an interrupt. If multiple interrupts occur simultaneously, the hardware resolves which interrupt is presented first. Following the presentation of each interrupt, a special hardware mask bit is reset to ensure that each interrupt is presented only once.

When the system responds to the interrupt, the current processor state is saved, and a device-specific interrupt handler is invoked. As part of that service, the interrupt source is

reset. When the device service is complete, an **end of interrupt** command is issued, which sets the special hardware mask, reenabling the presentation of interrupts on this level. If another interrupt is pending at this level, the EIS register in the system is set again.

Interrupt registers are shown in Figure 76. These registers are a protected system resource located in the IOCC address space between addresses X'–0 40 00 80' and X'–0 40 00 9F', and are only accessible to Load and Store instructions from the system processor when the Segment register privileged key is set to a value of 0. Attempts to access this address space when the privileged key is set to 1 results in a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

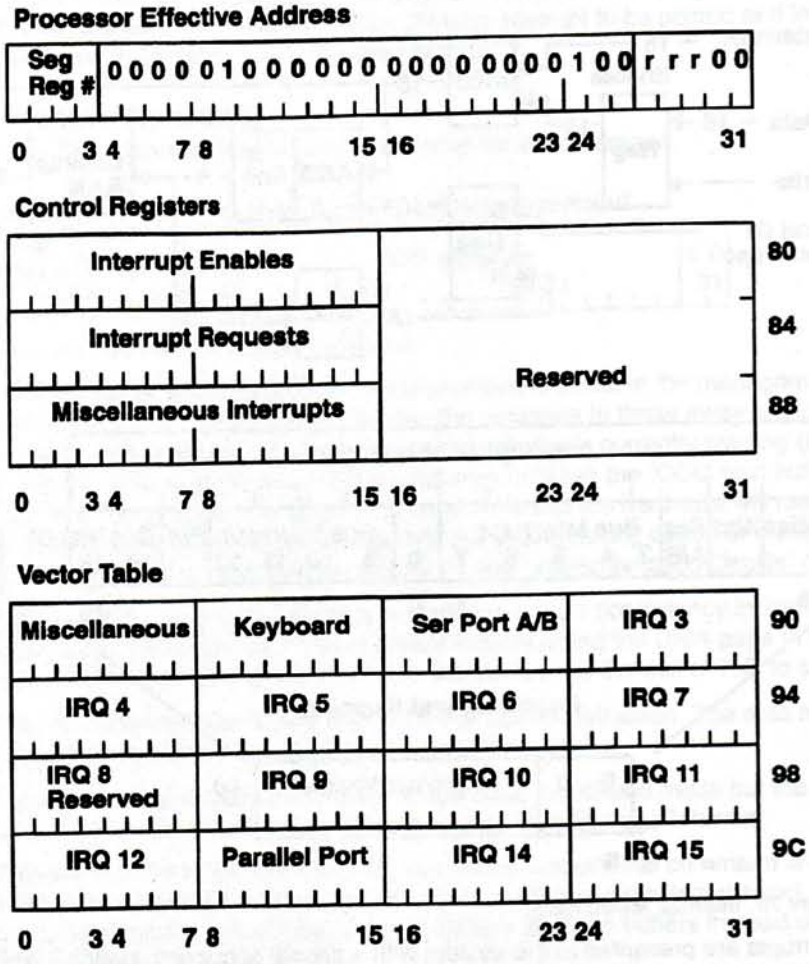


Figure 76. Interrupt Registers

- Register 80 – Interrupt Enable Register

  This register provides the ability to enable or disable any of the primary 16 interrupt requests. Bits 16 to 31 are reserved and should be set to a value of 0 on a Store instruction. On a Load instruction, bits 16 to 31 are indeterminate. No dynamic management of this register is necessary during interrupt service. It is provided primarily to allow disabling of unused, potentially noisy interrupts.

- Register 84 – Interrupt Request Register

  This register provides access to the device interrupt sources and can be read using an I/O Load instruction. Bits 16 to 31 are reserved and on a Load instruction are indeterminate. A Store instruction to this address is a NOP. A detailed description of each bit follows:

Bits	Description
0	Miscellaneous Interrupt: Miscellaneous interrupts are not directly vectored to the EIS register. The system unit provides one EIS register with 64 interrupts, of which the IOCC is allocated 16 levels. To fit within this maximum, the IOCC presents miscellaneous interrupts as a class interrupt, consuming one logical level. This appears in bit 0 (vector lookup 0), and is an OR of all the bits in register 88. If this interrupt is posted, the system is required to read IOCC register 88 to determine the cause of the interrupt. Bit 0 is set to a value of 1 when any miscellaneous interrupt occurs and bit 0 in the Enable register is set to a value of 1. This bit is a summary OR of register 88 and cannot be written. During an I/O Store instruction to this register, bit 0 is ignored. This bit is reset when register 88 is reset.
1	Keyboard Interrupt: This bit is set to a value of 1 when a keyboard interrupt occurs and bit 1 in the Enable register is set to a value of 1. This interrupt is level-sensitive and must be reset within the device prior to an interrupt return.
2	Serial Port Interrupts: This bit is set to a value of 1 when a board serial port 1 or serial port 2 interrupt occurs (Shared Interrupt) and bit 2 in the Enable register is set to a value of 1. This interrupt is level-sensitive and must be reset within the device prior to an interrupt return.
3–7, 9–12, 14–15	I/O Bus Interrupts: These bits are set to a value of 1 when I/O bus interrupts occur and their corresponding bits in the Enable register are set to a value of 1. These bits reflect the current signal level of each of the Micro Channel interrupt lines and are not latched. It is not necessary to reset these bits as part of interrupt service.
8	Reserved: This bit is reserved and must be set to a value of 0.
13	Parallel Port Interrupt: This bit is set to a value of 1 when a Standard I/O parallel port interrupt occurs and bit 13 in the Enable register is set to a value of 1. This interrupt is level-sensitive and must be reset within the device prior to an interrupt return.
16–31	Reserved: These bits are reserved and must be set to a value of 0. On a Load instruction, the value of bits 16 to 31 are indeterminate.

- Register 88 – Miscellaneous Interrupts Register

  The first two bits of this register contain IOCC errors not reported in the Channel Status registers. These errors are caused by asynchronous events or are associated with situations where no device interrupt is posted. As such, the IOCC reports these errors by way of its own interrupt.

  The third bit of this register provides an interrupt for the Standard I/O keyboard Ctrl-Alt-Anything sequence and is called a Keyboard External Interrupt.

  The summary OR of this register is presented as bit 0 of register 80.

This register is both read and written using I/O Load and Store instructions. Store instructions function only as a masked reset. Writing a value of 0 to a bit position resets that bit, while writing a value of 1 does nothing. A detailed description of each bit follows:

Bits	Description
0	Channel Check: This bit is set if the I/O bus 'chck' line is active during a Micro Channel operation (PIO or DMA slave) at the beginning of a cycle (after 'arb/gnt' signal falls and before the first time the 'cmd' signal falls). There should be no devices that asynchronously report errors by activating the 'chck' signal. However, if this occurs, the channel check posts an asynchronous IOCC error interrupt. Normally, in the system unit, the 'chck' signal is presented as a synchronous exception and a Data Storage interrupt is posted instead. Refer to "Exception Reporting and Handling" on page 2-85 and "Channel Check" on page 2-19 for more information.
1	Bus Timeout: This bit is set if an I/O bus timeout occurred. See "Bus Timeout" on page 2-20 for additional details. While this bit is active, the 'arb/gnt' signal is forced high, bus arbitration is suspended, and control of the I/O bus is unconditionally given to the IOCC.
2	Keyboard External: This bit is set when the Ctrl-Alt-Anything sequence is pressed at the Standard I/O keyboard and is called a Keyboard External Interrupt. It is presented to the system as an external interrupt. Software is then able to determine which key caused the interrupt and takes the appropriate action. This bit is implementation dependent. (See "Implementation Details" on page 2-86).
3–31	Reserved: These bits are reserved and must be set to a value of 0. On a Load instruction, the value of bits 16 to 31 are indeterminate.

- Register 90 to 9F – Vector Table

  This set of registers contains the interrupt vectors to be presented to the system EIS register. One vector is provided for each bit in register 84. The operating system loads this table with a set of 6-bit values corresponding to the interrupt priority desired.

  Note: The vector table is implementation-specific. (See "Implementation Details" on page 2-86.) Implementations that support a single I/O bus can fix the conversion of interrupt level to the EIS bit. This fixed conversion is the identify transform (that is, interrupt 0 to EIS bit 0, interrupt 5 to EIS bit 5, and so on.) When the vector table is not supported, a Load or Store instruction to the vector table addresses results in a Data Storage interrupt (invalid operation).

Special Facilities

Figure 77 shows the register organization within the IOCC. (For implementation details, see "Implementation Details" on page 2-86.)

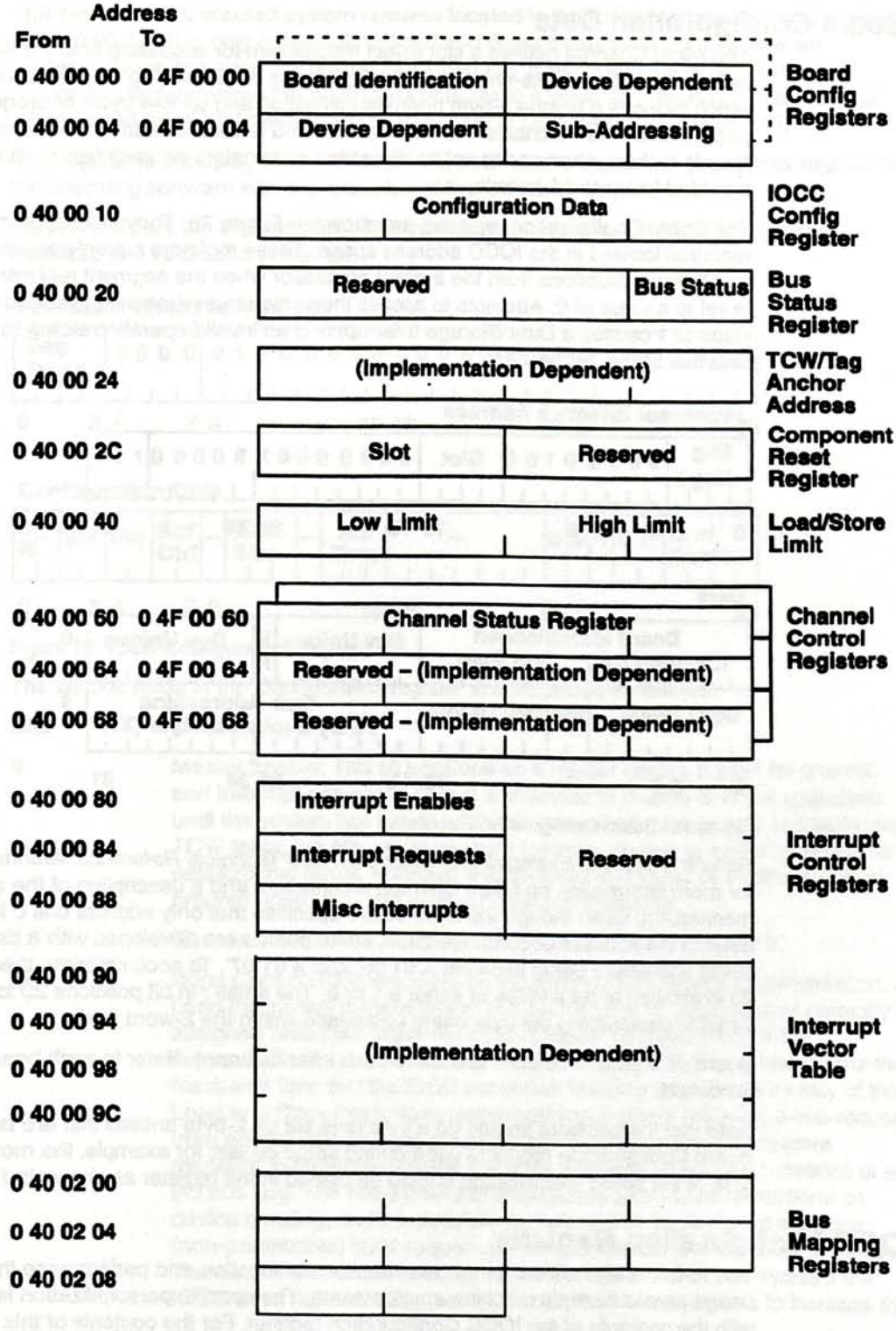


Figure 77. IOCC Registers

Board Configuration Data

The Micro Channel defines a slot select mechanism for accessing board-unique configuration data (byte-only access). Eight bytes of addressing are provided per board, which includes a unique 2-byte board identification and up to 4 bytes of programmable parameters. This mechanism is called setup, and is used at startup time to determine the boards in the system and to set configuration parameters on each board. Support is provided for up to 16 boards.

The Board Configuration registers are shown in Figure 78. They are a protected system resource located in the IOCC address space. These registers are only accessible to Load and Store instructions from the system processor when the segment register privileged key is set to a value of 0. Attempts to access these registers when the privileged key is set to a value of 1 causes a Data Storage Interrupt and an invalid operation status to be set in Channel Status register 15.

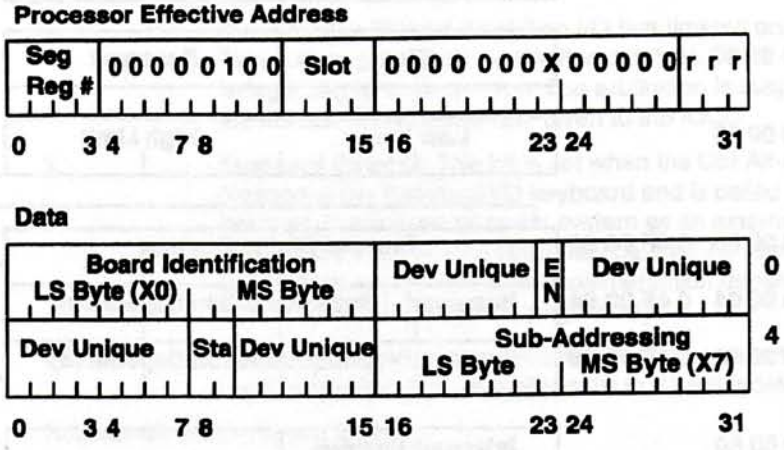


Figure 78. Board Configuration Registers

Refer to *Personal System/2 Hardware Interface Technical Reference: Architectures* manual for more information on Micro Channel architecture and a description of the setup mechanism. Even though the architecture specifies that only address bits 0 to 2 are to be used in the address decode operation, some boards are developed with a dependency on setup addresses being between X'01 00' and X'01 07'. To accommodate these boards, bit 23 is allowed to be a value of either a 1 or 0. The small *r* in bit positions 29 to 31 is a variable designating the byte being addressed within the 2-word field.

Board configuration data is unique to each specific board. Refer to each board specification for details.

Note that the software should do a byte reversal on 2-byte entities that are targeted for the Board Configuration registers used during setup cycles; for example, the most significant byte of the board identification should be placed in the register as shown in Figure 79.

IOCC Configuration Register

The IOCC design allows for certain variations of function and performance that optimize its usage across multiple machine environments. The specific personalization is established with the contents of the IOCC Configuration register. For the contents of this register for specific implementations, see "Implementation Details" on page 2-86.

This register is a protected system resource located in the IOCC address space at address X'–0 40 00 10'. It is only accessible to Load and Store instructions from the system processor when the segment register privileged key is set to a value of 0. Attempts to access this register when the privileged key is set to a value of 1 result in a Data Storage Interrupt and an invalid operation error status set in Channel Status register 15.

This register is set up by hardware and ROM code and is treated as a read-only register by the operating software with the exception of the master enable bit.

Figure 79 shows the organization of the configuration register. Bit 0 in this register is initialized to a value of 0 at startup.

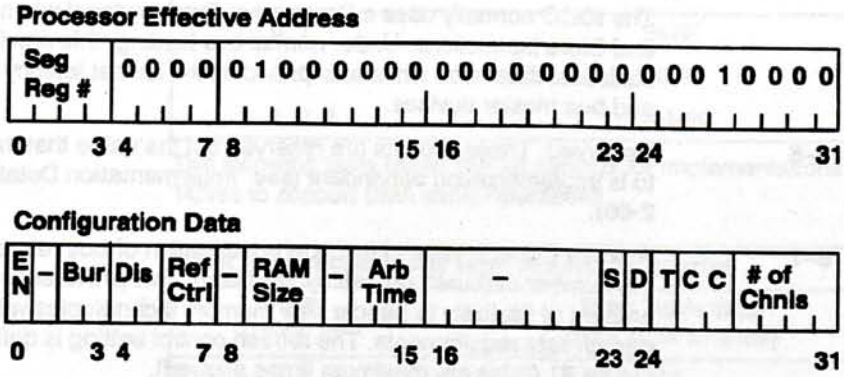


Figure 79. IOCC Configuration Register

The various fields in the Configuration register are described as follows:

Bits	Description
0	Master Enable: This bit functions as a master enable control for channel and interrupt operations only. It is intended to disable channel operations until the system has initialized the Channel Control registers, tag table, and TCW table, but also could be used following startup to assist recovery from catastrophic errors. Normally, this bit is set to a value of 1 following initial program load (IPL) and is never changed thereafter.
1	Reserved: This bit is reserved and must be set to a value of 0.
2–3	Burst Control: Programmable burst control is an optional implementation. A Load instruction to these bits indicates the state implemented or currently assigned (see also "Implementation Details" on page 2-86). If not supported, a Store instruction to these bits is a NOP. These bits control the maximum time that the IOCC continues to utilize the I/O bus by way of the Load and Store instructions under bursting protocol following a bus request from another device. This set of controls is provided as a protective measure to retain reasonable interrupt response time in the presence of an I/O bus hog. The Micro Channel architecture places few restrictions on device bursting, and it is possible for a device to be designed with long (non-preemptive) burst sequences, even if operating in the fairness mode. The device then receives a disproportionate number of bus cycles if the IOCC does not also utilize non-preemptive burst sequences to increase the

blocking factor. It is the responsibility of the IOCC to ensure that the 7.8-microsecond bus timeout constraint is adhered to.

2	3	
0	0	**Complete Current Cycle**
0	1	**1.6 microsecond**
1	0	**3.2 microsecond**
1	1	**6.4 microsecond**

Figure 80. Bit 2 and 3 Burst Control Setting

The IOCC normally uses a Preemptive Burst protocol when executing Load and Store instructions. Under normal bus loading, this provides high statistical data rates while also providing the lowest latency to DMA slave and bus master devices.

4–5 Reserved: These two bits are reserved but the value that they must be set to is implementation dependent (see "Implementation Details" on page 2-86).

6–7 Refresh Control: These bits allow specification of bus refresh periodicity and the number of (burst) refresh cycles taken. This provides for a certain amount of flexibility to handle new memory technologies with different refresh rate requirements. The refresh control setting is defined as shown in Figure 81 (rates are maximum times allowed).

6	7	Rate	# Cycles
0	0	**Off**	–
0	1	**60 microsecond**	**4**
1	0	**30 microsecond**	**4**
1	1	**15 microsecond**	**4**

Figure 81. Refresh Control Setting

8 Reserved: This bit is reserved and must be set to a value of 0.

9–11 TCW Table Size Specification: These bits allow specification of the amount of control RAM (TCW and Tag) to be packaged with the IOCC. Different applications require different amounts of TCW table, and the IOCC design allows this size to be varied. This provides the flexibility to optimize cost and function across a wide range of system applications. These bits should be personalized to match the size of the RAM provided with the IOCC (in terms of the number of TCWs supported). The **TCW Table Sizes for**

Combination TCW and Tag table shows the bit settings for implementations where tags are used for DMA slave operations.

TCW Table Sizes for Combination TCW and Tag	
Bit 9 10 11	**TCW Table Size (# of TCW entries)**
0 0 0	24K
0 0 1	56K
0 1 0	120K
0 1 1	248K
1 0 0	504K
1 0 1	1016K

The following table shows the bit settings for implementations that use TCWs to support DMA slave operations.

TCW Table Sizes When Tags Are Not Supported	
Bit 9 10 11	**TCW Table Size (# of TCW entries)**
0 0 0	8K
0 0 1	16K
0 1 0	32K
0 1 1	64K
1 0 0	128K
1 0 1	256K
1 1 0	512K
1 1 1	1024K

The Tag table has 4096 entries, and the remainder of the RAM is allocated to the TCW table. If both the DMA slave and the bus master operations are handled using TCWs, all of the RAM is available for the TCW table. Due to the mapping of bus I/O and bus memory into one address space, no bus memory is allowed between 0 and 64K bytes, and the first 16 TCW entries are never accessed.

12 Reserved: This bit is reserved and must be set to a value of 0.

Bits	Description
13–15	Arbitration Time: These bits allow specification of the arbitration time on the Micro Channel. Different systems applications have different bus configurations and loading, and require different arbitration values. These values can be varied from the architected minimum to a value greater than that provided by the RT system bus application. Each arbitration value in the Arbitration Time Configurations table represents a range, for example, 100 nanoseconds equals 100 to 200 nanoseconds.

Arbitration Time Configurations

Bits 13 14 15	Arbitration Time (nanoseconds)
0 0 0	100
0 0 1	200
0 1 0	300
0 1 1	400
1 0 0	500
1 0 1	600
1 1 0	700
1 1 1	800

16–22	Reserved: These bits are reserved and should be a value of 0.
23	TCW and Tag Tables in System Memory: A value of 1 in this bit indicates that the TCW and tag tables are in system memory. The register for anchoring the address of a system memory based TCW and tag table is at X'–0 40 00 24'. All pages in system memory provided for TCW and tag tables are continuous in real memory and permanently pinned. The TCW and tag tables are only accessed through the IOCC space and are not mapped into the PFT. Any error while accessing this memory results in a TCW and Tag access error. This area is not scrubbed. A value of 0 in this bit indicates that nonsystem memory is used for the TCW and tag tables.
24	Dual Buffer Support and Bus Mapping Register: This bit indicates whether or not the dual buffering and Bus Mapping register option of the architecture is supported. A value of 1 in this bit indicates that the dual buffer and Bus Mapping register option of the architecture is supported. A value of 0 in this bit indicates that it is not supported. For implementation details, see "Implementation Details" on page 2-86.
25	DMA Slave TCW or Tag Bit: This bit indicates whether the DMA supports the use of tags or TCWs for DMA slave operations. A value of 0 indicates tags are supported.

Bits	Description
26–27	Cache Buffer Support and Cache Coherency: These bits have the following meanings:

26	27	
0	0	**Buffered Mode, Software Enforced Consistency**
0	1	**Unbuffered Mode**
1	0	**Reserved**
1	1	**Reserved**

Figure 82. Cache Mode Bits

In the buffered mode, the IOCC buffers exist, and PIOs to system memory are allowed. In the unbuffered mode, there are no IOCC buffers and PIOs to system memory are not allowed. See "Maintaining Consistency" on page 2-36.

28–31 Number of DMA Slave Channels: These bits indicate the number of DMA slave channels (that is, the number of DMA Slave Control registers) that are supported. Both B'0000' and B'1111' indicate that 15 channels are supported. Also, B'0001', B'0010', B'0011' indicate that one, two, and three channels are supported, respectively. The number of channels supported is implementation-specific. However, the number of arbitration levels supported is not implementation-dependent, and must be equal to 16. (See "Implementation Details" on page 2-86). If the implementation supports tags, then all 15 DMA slave channels must be supported. The minimum required by the Micro Channel architecture is 2. The minimum required by the system architecture is the number of slots plus the number required by the Standard I/O devices. If buffers are supported, the number of buffers must equal the number of channels supported.

Bus Status Register

The Bus Status register (BSR) is a diagnostic facility that aids in I/O error isolation. It is comprised of one R/W register and provides the ability to set and sample signals on the I/O bus.

The BSR is a protected system resource located in the IOCC address space at address X'–0 40 00 20'. It is only accessible to Load and Store instructions from the system processor when the segment register privileged key is set to a value of 0. Attempts to access these registers when the privileged key is set to a value of 1 causes a Data Storage

Interrupt and an invalid operation error status to be set in Channel Status register 15. Figure 83 shows the Bus Status register.

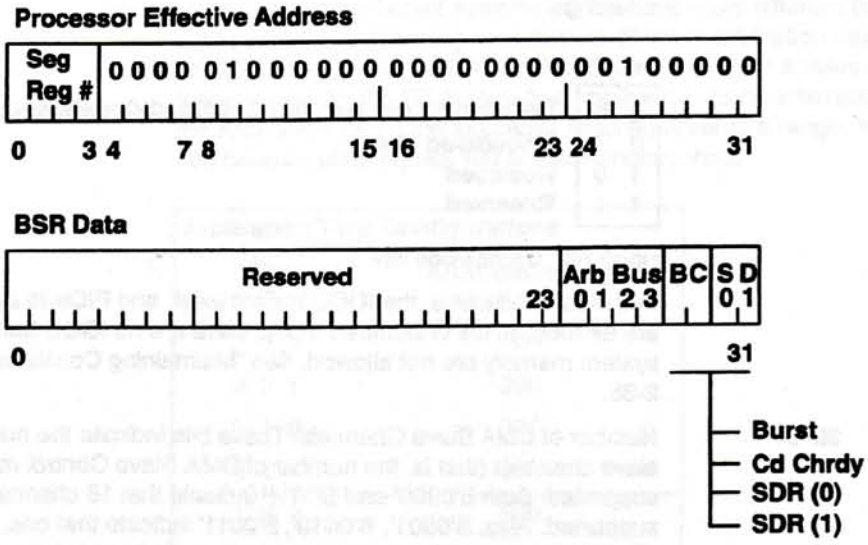


Figure 83. Bus Status Register

The 'arb' bus lines, 'burst' signal, 'cd chrdy' signal, and 'sdr (0)' and 'sdr (1)' signals are latched in the BSR latches when a bus timeout error occurs. The 'arb' bus bit 0 is the least significant and bit 3 is the most significant bit. If a bus timeout error occurs during an I/O cycle, further bus errors will not be trapped until the error interrupt is cleared out of the Miscellaneous Interrupt register. As such, the BSR contains a copy of the sampled I/O bus signal lines at the time of the first error. No provision is made for saving bus states for successive errors.

Results of a Store instruction are implementation-dependent (see "Implementation Details" on page 2-86) On a Load instruction, the data returned is the contents of the register as described, if an error has occurred (bit 1 of the Miscellaneous Interrupt register is on); the contents of bits 0 to 23 are indeterminate.

TCW and Tag Anchor Address Register

This register specifies the starting address of the TCW and tag table when that table is in system memory (as indicated by bit 23 of the IOCC Configuration register). This register is undefined when bit 23 of the IOCC Configuration register is a 0, and a Store instruction to this register when bit 23 is a 0 will cause a Data Storage Interrupt, and an invalid operation status to be set in Channel Status register 15.

The TCW and Tag Anchor Address register is a protected system resource located in the IOCC address space at address X'–0 40 00 24'. It is only accessible to Load and Store instructions from the system processor when the Segment register privileged key is set to a value of 0. Attempts to access this register when the privileged key is set to a value of 1

causes a Data Storage Interrupt and invalid operation status to be set in Channel Status register 15. Figure 84 shows the TCW and Tag Anchor Address register.

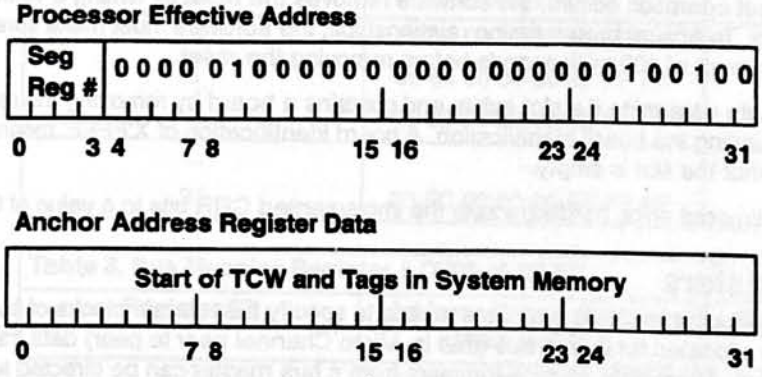


Figure 84. TCW and Tag Anchor Address Register

Software must guarantee that the table starting address is on a boundary which is equal to the size of the table. For example, for a 128K-byte table must start on a 128K byte boundary.

Component Reset Register

The Component Reset register (CRR) is comprised of one register and provides the ability to individually drive the resets to each I/O slot. Writing a value of 0 into a bit position resets that slot, and writing a value of 1 removes the reset.

The CRR is a protected system resource located in the IOCC address space at the address X'–0 40 00 2C'. It is accessible to Load and Store instructions from the system processor when the segment register privileged key is set to a value of 0. Attempts to store into this register when the privileged key is set to a value of 1 causes a Data Storage Interrupt and an invalid operation error status to be set in Channel Status register 15.

Figure 85 shows the Component Reset register. The actual number of slots supported is *implementation dependent and is consistent* with the IOCC configuration definition. The use of the reserved bits is also implementation dependent, and *the usage includes resets for the* Standard I/O devices. On a Load instruction to this register, the value of the unused bits is implementation dependent. See "Implementation Details" on page 2-86.

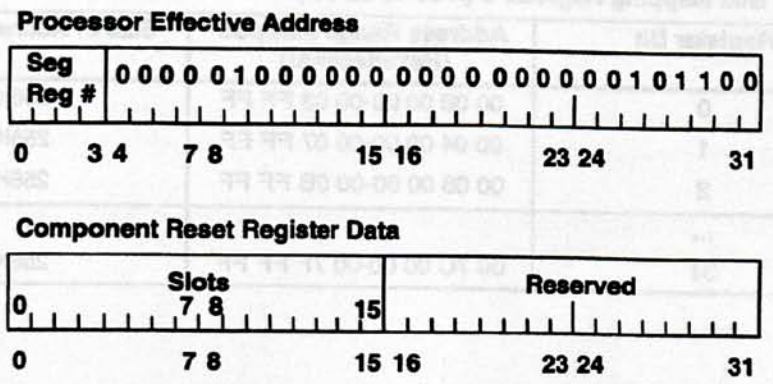


Figure 85. Component Reset Register

The CRR is initialized to a value of 0 at startup. This sets and holds a bus reset to all the I/O boards until explicitly enabled by a startup diagnostic utility.

After a reset operation occurs, the software removes the reset by writing a value of 1 to the board slots. To ensure proper timing relationships, the software must make sure the reset is held a minimum of 100 milliseconds before removing the reset.

Software can determine if a slot exists and contains a board by removing the reset to the slot and reading the board identification. A board identification of X'FFFF' means that no slot exists, or that the slot is empty.

On a bus timeout error, hardware sets the implemented CRR bits to a value of 0.

Bus Mapping Registers

The Bus Mapping registers provide a means to specify that certain blocks of bus address space are allocated for bus to bus (that is, Micro Channel peer to peer) data transfers by bus masters. Alternately, all data transfers from a bus master can be directed to the bus address space by setting bit 2 of that bus master's CSR to a value of 0. These registers allow for the flexibility of directing some of a bus masters transfers to the bus memory and some to system memory without having software intervene to change the setting of CSR bit 2 for that bus master. The Bus Mapping registers are an optional feature of the architecture. Their presence is indicated by bit 24 in the IOCC Configuration register being set to a value of 1.

The following Tables 1, 2, and 3, show the address ranges mapped by each bit of each Bus Mapping register. If a bus master has its CSR bit 2 (mapping bit) set to a value of 1 and a bit in the Bus Mapping registers is set to a value of 0, then the corresponding range of bus address space will NOT be mapped to system memory for that bus master (that is, a bus master access to this range will result in a bus to bus transfer cycle). If a bus master has its CSR bit 2 (mapping bit) set to a value of 1 and a bit in the Bus Mapping registers is set to a value of 1, then the corresponding range of bus address space is mapped to system memory for that bus master. A bus master whose CSR bit 2 is set to a value of 0 will always be accessing the bus address space (doing a bus-to-bus operation), regardless of the setting of the corresponding Bus Mapping register bit (that is, the cycle is a bus to bus cycle if either the CSR bit 2 of the bus master doing the access or the corresponding bit of the Bus Mapping register is set to a value of 0). Notice that there are three granularities of the mapping depending on the address range mapped.

Table 1. Bus Mapping Register 0 (X'00 40 02 00')

Register Bit	Address Range Mapped (Hexadecimal)	Size of Address Range
0	00 00 00 00-00 03 FF FF	256K
1	00 04 00 00-00 07 FF FF	256K
2	00 08 00 00-00 0B FF FF	256K
...	...	...
31	00 7C 00 00-00 7F FF FF	256K

Table 2. Bus Mapping Register 4 (X'00 40 02 04')		
Register Bit	**Address Range Mapped (Hexadecimal)**	**Size of Address Range**
0	00 80 00 00-00 83 FF FF	256K
1	00 84 00 00-00 87 FF FF	256K
2	00 88 00 00-00 8B FF FF	256K
...	...	...
31	00 FC 00 00-00 FF FF FF	256K

Table 3. Bus Mapping Register 8 (X'00 40 02 08')		
Register Bit	**Address Range Mapped (Hexadecimal)**	**Size of Address Range**
0	01 00 00 00-03 FF FF FF	48M
1	04 00 00 00-07 FF FF FF	64M
2	08 00 00 00-0B FF FF FF	64M
...	...	...
15	40 00 00 00-43 FF FF FF	64M

These registers are protected system resources located in the IOCC address space at the address X'00 40 02 00' to X'00 40 02 08'. They are accessible to Load and Store instructions from the system processor when the segment register privileged key is set to a value of 0. Attempts to store into these registers with the privileged key is set to a value of 1 will cause a Data Storage Interrupted and an invlaid operation error status to be set in Channel Status register 15.

System I/O and Standard I/O

Two classes of devices are described in this section, the System I/O and the Standard I/O.

System I/O is defined as facilities in the I/O space intrinsic to the system but not normally considered I/O devices. Included in this category are NVRAM, clock and calendar, operator panel, system registers, and on card sequencers (OCS). System I/O, though in the I/O space, is isolated from the I/O bus by way of an internal bus and is a protected resource.

Standard I/O devices in the system unit are defined as those I/O devices intrinsic to a basic workstation, and as such, are included as part of the base machine. These devices do not necessarily occupy feature slots because these devices are not optional features. The list of items which fall into this category is implementation specific (see "Implementation Details" on page 2-86).

System I/O

System I/O is located in the IOCC control space, is privileged, and is only accessible when the segment register privileged bit is set to a value of 0. Attempts to access this address space when the privileged bit is set to a value of 1 causes a Data Storage Interrupt to be posted and an invalid operation error status to be set in Channel Status register 15. The remainder of this section contains information describing System I/O.

System Registers

System registers are located in the IOCC control space between the addresses X'–0 40 00 C0' and X'–0 40 00 FF' defining a contiguous space of 64 bytes. These registers are implementation-dependent (see "Implementation Details" on page 2-86).

Nonvolatile RAM

The Nonvolatile Random Access Memory (NVRAM) is located in the IOCC control space between X'–0 A0 00 00' and X'–0 BF FF FF' and occupies 2M-bytes of address space. The amount of NVRAM in the system is implementation-specific (see "Implementation Details" on page 2-86).

Standard I/O

The Micro Channel provides for a 16-bit bus I/O address. To access a device within this address space, effective address bits 4 to 15 and segment register bits 28 to 31 must all be a value of 0.

Accesses to the I/O bus are checked for proper access authority by way of an address range check, restricting user programs to access only authorized devices. However, since the IOCC cannot intercept or stop accesses to bus attached memory or bus I/O devices by a bus master on the I/O bus, no access checking is performed when a bus master addresses these devices.

Actual Standard I/O address assignment are implementation dependent (see "Implementation Details" on page 2-86).

Exception Reporting and Handling

Refer to *Personal System/2 Hardware Interface Technical Reference: Architectures* manual for more information on Micro Channel architecture and for definitions of the data and address parity on the Micro Channel.

The following are general guidelines that were followed in designing the system units and adapters, and should be followed in designing new adapter boards for the machines:

- Full parity support is recommended for all address and data buses for all adapter boards, internal boards, and internal devices (such as Standard I/O devices, NVRAM, and System registers). Full address and data parity support is defined as traversing the complete paths of the address and data buses (generate parity at the signal source and check parity at each destination point where the address and data will be used).
- Internal boards (Standard I/O and I/O Boards) provide both address and data parity support to each of their devices.
- Adapter boards to be supported for system units should provide both address and data parity support at the board connector and on all *internal* data and address buses.
  - 8- and 16-bit devices should provide the 32 bit board connector to gain access to all the required parity signals.
  - 8- and 16-bit devices, should also implement a notch in the board tab so they can be installed in a 16-bit board slot.

    Note: Suitable pull-up resisters should be utilized as appropriate.
- Adapters that do not use the 32-bit board connector (8- and 16-bit data), should support data parity as a minimum requirement. The objective is to include the 32-bit connector described previously to allow address parity, also, if possible.
- Devices and boards should meet the signal timing specifications described in the Micro Channel architecture documents. For Micro Channel architecture information, refer to the *Personal System/2 Hardware Interface Technical Reference: Architectures* manual.

Implementation Details

This section provides implementation details for system Models 320, 32E, 32H, 520, 52H, 530, 530E, 53H, 540, 550, 550E, 550S, 560, 560F, 730, 930, 950 and 950E.

Implementation details for other models can be found in the Input/Output (I/O) architecture implementation details sections of the product-specific technical information manuals.

Streaming Data Protocol

These models support the 4-byte Streaming Data protocol.

Board Configuration Register

Figure 78 on page 2-74 shows the board configuration register assignments.

IOCC Configuration Register

Some of the bits in the IOCC Configuration register indicate support or nonsupport of various implementation-dependent features. The following is a summary of the definition of the IOCC Configuration register implementation for these models. In the case of read-only memory (ROM) code initialized bits, the value that the ROM must initialize these bits to is shown. For the bits of the IOCC Configuration register that are not documented in the following descriptions, the ROM code must initialize those bits to a value of 0.

Bits	Description
2–3	Burst Control: These models support the programmable burst control in bits 2 and 3 of the IOCC Configuration register. These bits are set to B'11' (6.4 microsecond) by the ROM code.
4–5	Reserved: These bits are reserved and must be set to B'01'. Reserved bits are set to B'01' by the ROM code.
6–7	Refresh Control: These bits are set to B'01' (60 microseconds refresh) by the ROM code.
9–11	TCW Table Size Specification: These bits are set to B'010' by the ROM code.
13–15	Arbitration Time: These bits are set to B'011' (400 nanoseconds) by the ROM code.
23	TCW and Tag Tables in System Memory: These models support nonsystem memory for TCW and tag tables as indicated by a 0 in this bit.
24	Dual Buffer and Bus Mapping Register Support: These models do not support the dual buffer and Bus Mapping register option of the architecture, as indicated by a 0 in this bit.
25	DMA Slave TCW and Tag: These models support the use of tags for DMA slave operations as indicated by a 0 in this bit.
26–27	Buffer Support and Coherency: These models support the use of buffers for bus master and DMA slave operations that are managed by software, as indicated by a B'00' in these bits. This also indicates that PIO operations to system memory are supported.
28–31	Number of DMA Slave Channels: These models support the use of 15 channels for DMA slave operations as indicated by B'0000' in these bits.

System Registers

Figure 86 shows the register assignments within this area.

Software polls the Power Status and Keylock Decode register (address X'0 40 00 E4') to determine if any bit within that register changes state, and then tests to determine the bit that caused the state change in order to take the proper action. Bits 28 to 31 in this register are the cover keylock switch-position decode bits and are used by ROM and software to determine proper IPL procedures based on the switch position. (The keyboard lock on these models is a software function.)

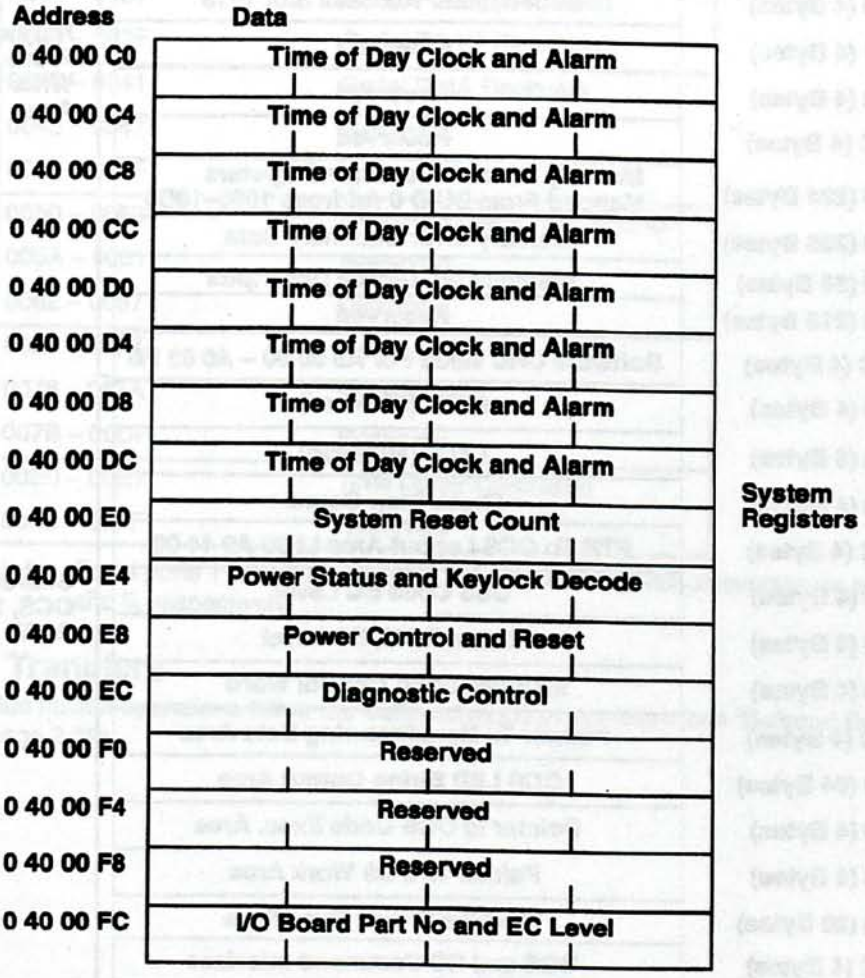
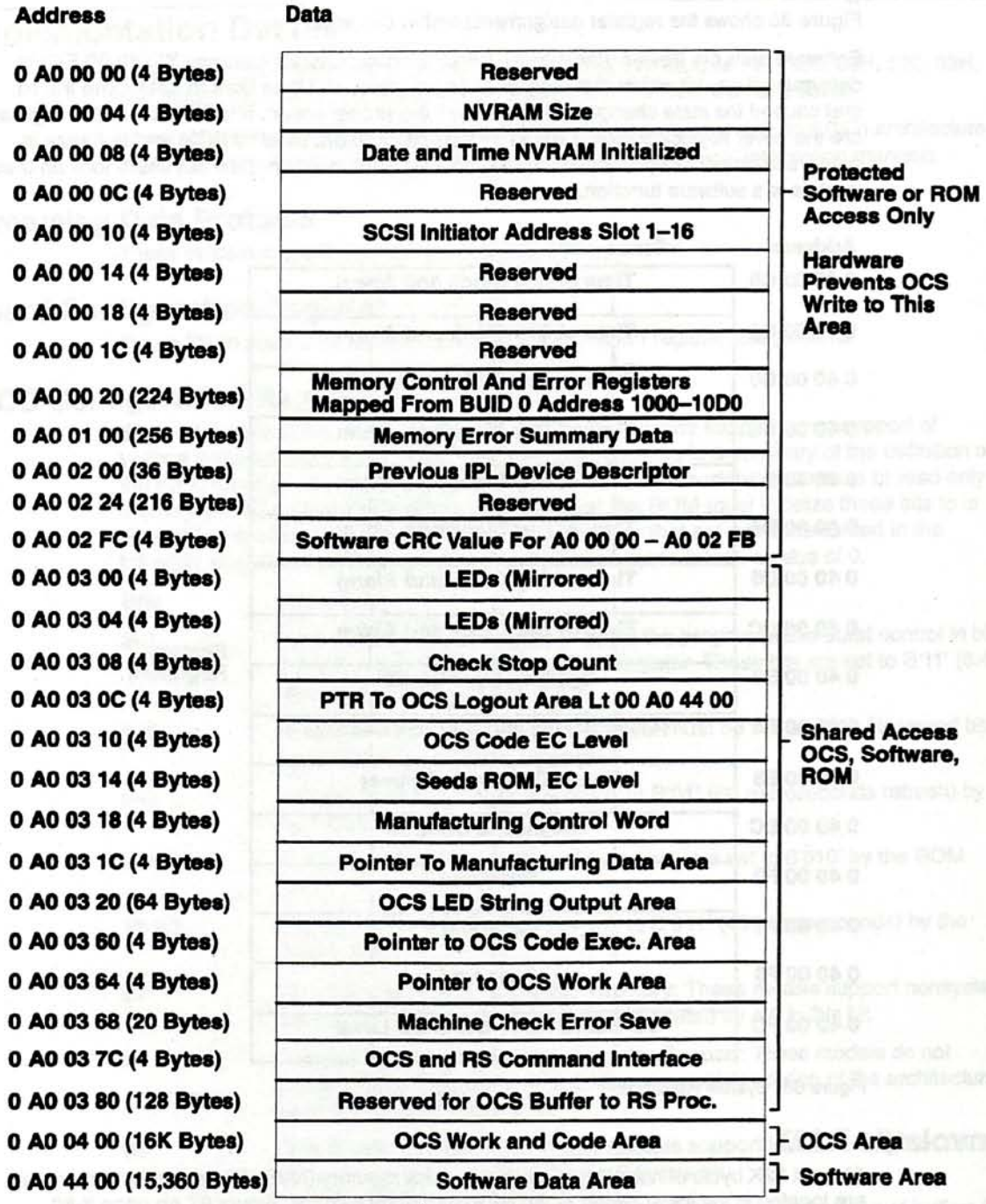


Figure 86. System Registers

Nonvolatile RAM

At least 32K bytes of nonvolatile random access memory (NVRAM) are implemented and are located in the lower range of the NVRAM address space. Figure 87 on page 2-88 shows the address assignments for the NVRAM area.



Note: **For systems with greater than 32K bytes of Nonvolatile RAM, the extra RAM increases the software data area.**

Figure 87. NVRAM Addressing

Standard I/O

The Standard I/O Address Map table shows a Standard I/O address map indicating the address assignments for each Standard I/O device.

Standard I/O Address Map	
Hex Address Range	**Standard I/O Device**
0000 – 002F	Reserved
0030 – 0037	Serial Port 1 (See note)
0038 – 003F	Serial Port 2 (See note)
0040 – 0041	Serial DMA Registers
0042 – 0047	Reserved
0048 – 004F	Mouse
0050 – 0059	Keyboard, Tablet and Sound
005A – 0061	Reserved
0062 – 0067	Diskette
0068 – 0077	Reserved
0078 – 007A	Parallel Port
007B – 00DF	Reserved
00E0 – 00E7	Time Delay Command
00E8 – 00FF	Reserved

Note: Serial ports 1 and 2 are referred to in the software documentation as serial ports A and B, respectively.

Bus Master Transfers

Bus master operations follow the buffered mode of operation (see "Buffered Bus Master" on page 2-39).

Component Reset Register

Up to eight slots plus the Standard I/O are supported. Bits 0 to 7 of this register represent the eight slots. Bit 31 is for the Standard I/O. On a Load instruction, the value of bits 8 to 30 are indeterminate. The CRR and Board Configuration Register Assignments table shows the logical slot number (Component Reset register bit) for the devices.

CRR and Board Configuration Register Assignments			
Logical Slot Number (CRR bit Number)	**Board Configuration Register Slot Number**	**Physical Slot Number**	**Comments**
0	0	1	
1	1	2	
2	2	3	
3	3	4	
4	4	5	Not used in 4-slot models
5	5	6	Not used in 4-slot models
6	6	7	For 4-slot models, used for the Direct Bus Attached file
7	7	8	For 4-slot models, used for the Direct Bus Attached file
8 to 30		not used	
31	X'F'	Standard I/O	

Notes on Error Detection

- IOCC and I/O bus protocol errors are not logged in the Channel Status register.
- TCW errors are parity errors, not ECC errors.

Bus Timeout

The time period is the time between refresh cycles (which is programmable through bits 6 and 7 of the IOCC Configuration register; see "IOCC Configuration Register" on page 2-74) plus the amount of time the device was on the bus prior to the first refresh cycle. For example, for a 15 microsecond refresh, the time range would be 15 to 30 microseconds, and for a 60 microsecond refresh, the time range would be 60 to 120 microseconds.

I/O Interrupts

The coded method of handling I/O interrupts is supported, including the use of the interrupt vector tables.

Power-On Reset

A power-on reset, system reset, or bus timeout, resets the master enable bit in the Configuration register. When this bit is a value of 0, the following is accomplished:

- The 'preempt' signal is de-gated, disabling channel arbitration.
- Interrupt presentation is inhibited to the system.

Also, on power-on reset, system reset, or bus timeout, the following is accomplished:

- The Component Reset register is reset.
- A reset condition is forced to all I/O slots.

The master enable bit can be set or reset by a Store instruction to the IOCC Configuration register. Figure 88 shows the system implementation.

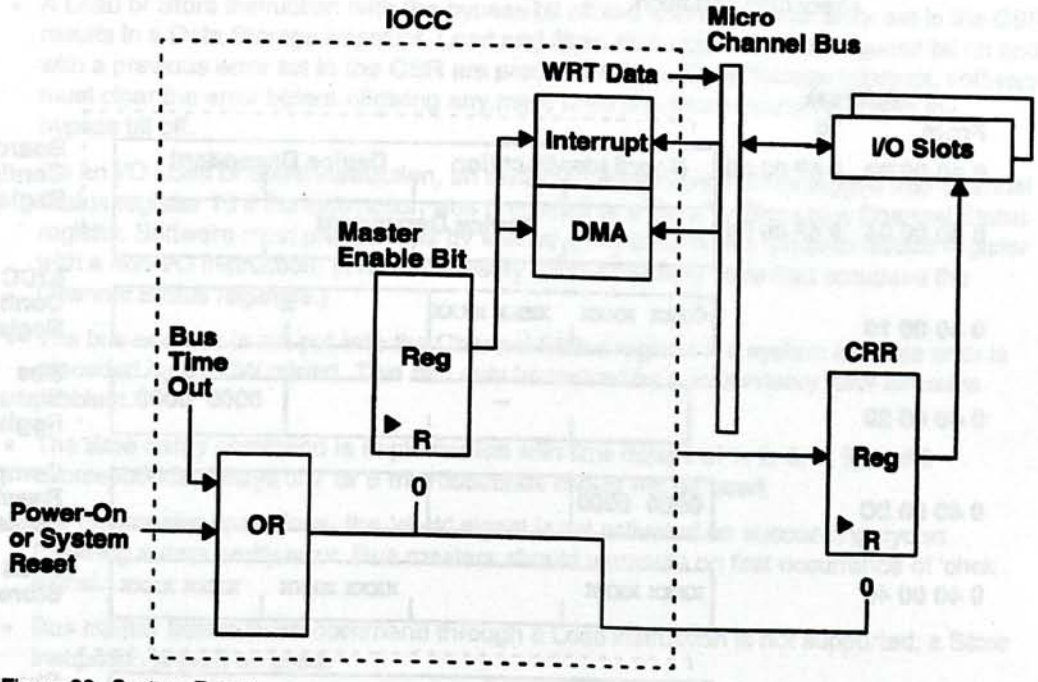


Figure 88. System Reset

IPL Procedures

Figure 89 on page 2-92 shows the power-on state of the IOCC registers. Indeterminate power-on states are indicated with an *x*, and undefined states are indicated with a dash (–). Attempts to read an IOCC register with an *x* before it has been initialized can result in a parity error, and the IOCC error interrupt mask should be disabled. The Channel Control registers and the interrupt vector table must be initialized with the Store instruction to establish good parity in these registers.

The TCW table, tag table, and IOCC memory also turn on in an indeterminate state. Attempts to read these address spaces before they have been initialized can result in parity errors, and the IOCC error interrupt mask should be disabled until after these spaces are initialized. These facilities must be initialized with a sequence of Store instructions to establish good parity.

Hardware provides a means for ROM to set the buffers and registers in the appropriate invalid state at power-on. Following a power-on condition, the following procedure must be followed to initialize the IOCC:

1. Initialize the IOCC Configuration register.
2. Reset the Interrupt Control registers.
3. Initialize the Channel Control registers, register 8 bit 2(I) to a value of 1, all other bits to a value of 0. Register 0 and 4 should be reset to a value of 0.
4. Reset the Load and Store Limit registers.

5. Initialize the interrupt vector table.
6. Initialize the TCW table.
7. Initialize the tag table.

Except for the master enable bit being reset, the IOCC does not lose any state information following a check stop reset. Thus, it is not necessary to reinitialize the IOCC following a check stop condition.

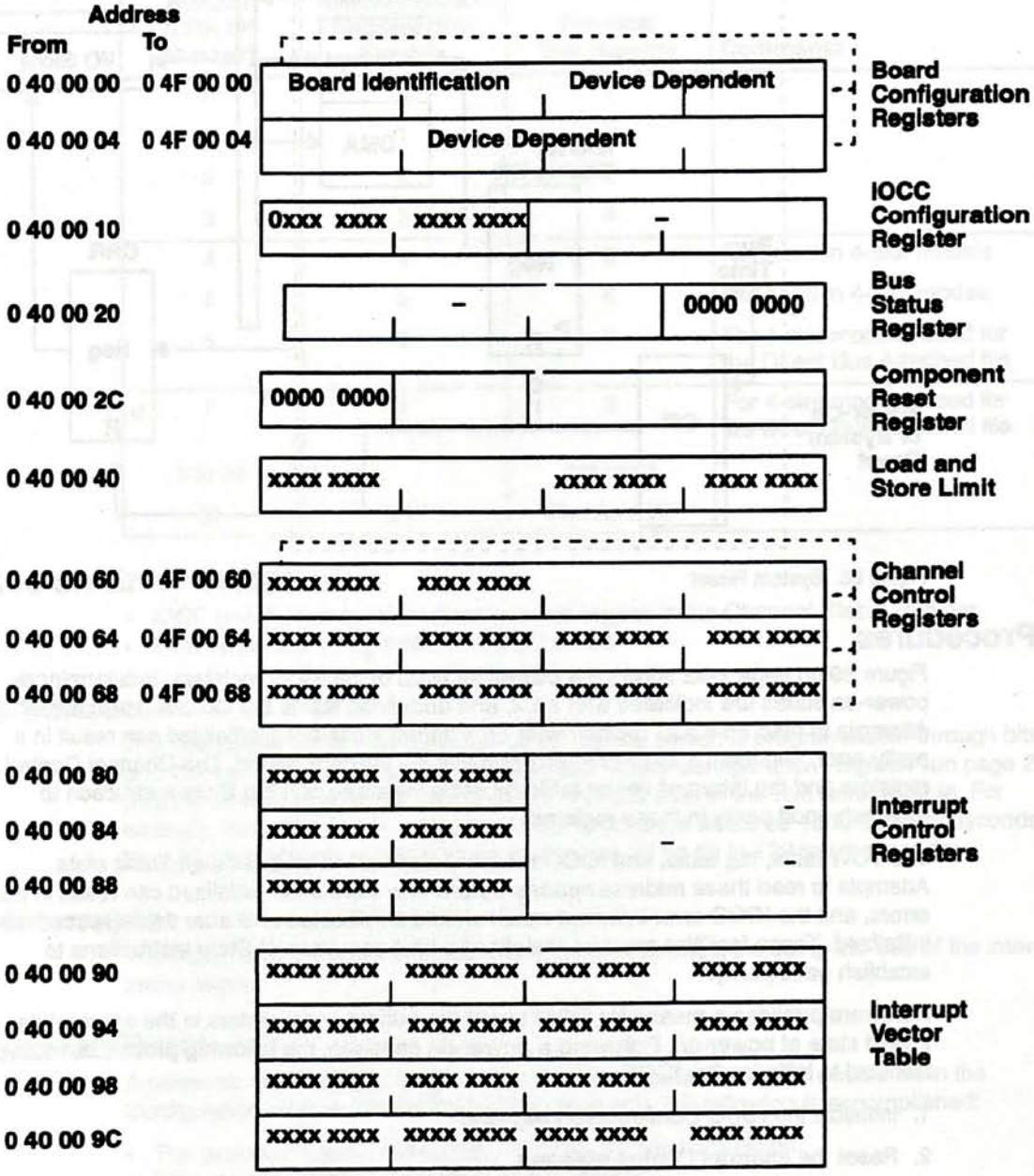


Figure 89. IOCC Power-On States

Deviations from the I/O Architecture

The following deviations are specific to system Models 320, 32E, 32H, 520, 52H, 530, 530E, 53H, 540, 550, 550E, 550S, 560, 560F, 730, 930, 950 and 950E. It has been verified that these systems, with these stated deviations, function satisfactorily. While this listing reflects good faith reasonable effort, no representation or guarantee is made that this listing is exhaustive.

- A Load or Store instruction with the bypass bit off and with a previous error set in the CSR results in a Data Storage interrupt. Load and Store instructions with the bypass bit on and with a previous error set in the CSR are processed. On a Data Storage interrupt, software must clear the error before allowing any more Load and Store instructions with the bypass bit off.
- On an I/O Load or Store instruction, an Invalid Operation error is not logged into Channel Status register 15 if the instruction was preceded by a Load or Store to a Channel Status register. Software must prevent this by following any access to a Channel Status register with a non-I/O instruction. (The supervisory code is the only code that accesses the Channel Status registers.)
- The bus address is not put into the Channel Status register if a system address error is preceded by a TCW reload. This can only be caused by a supervisory level software problem.
- The **time delay** command is implemented with time delays of 1, 2, 3, 4, 5, and 6 microseconds; delays of 7 or 8 microseconds should not be used.
- For bus master operations, the 'chck' signal is not activated on succeeding cycles following a data parity error. Bus masters should terminate on first occurrence of 'chck' signal.
- Bus master **buffer flush** command through a Load instruction is not supported; a Store instruction should be used.
- The Streaming Data protocol is not supported for IOCC initiated Load or Store, and DMA Slave operations.

Chapter 3. Vital Product Data

Chapter Contents

Description

Vital product data (VPD) uniquely defines each hardware, software, and microcode element of a system. Configuration data identifies the physical and logical location of each hardware element of a system including addressing information. The combination of configuration and VPD provides the system with a bill of material description that typically includes the assembly part number, Engineering Change (EC) level, serial number, and other detailed information. The objective from a system point of view is to determine this information by reading this data directly from the hardware, software, and microcode components.

Note: This chapter provides information for system models 32x, 34x, 35x, 36x, 52x, 53x, 540, 55x, 56x, 58x, 730, 930, 95x, 97x, and 98x. Information for other system models can be found in the product-specific technical information manual for those models.

Certain information such as machine type, model and external serial number (for example, deskside system numbers) is not in machine-readable form. This information is provided in Nonvolatile Random Access Memory (NVRAM) during manufacturing. Access to configuration and VPD information is provided by the Operating System with the System Management Interface Tools. This interface allows the user to add VPD (such as a serial number) as well as other user information such as owner, physical location, and information applicable to inventory or asset control.

Importance

The collection of configuration and VPD offers the following advantages:

- Assists the operating system in auto-configuring the system and its components.
- Assists diagnostics in problem determination and fault isolation:
  - Error logging includes VPD information so that a historical entry is associated with a serialized unit (such as an adapter).
  - Identifying the physical and logical location of failing units for replacement.
- Assists the operating system in determining the proper device driver and loadable microcode level.
- Assists the user in maintaining asset and inventory control.
- Provides a means of licensing software on a processor ID or serial number basis.

Characteristics

Configuration and VPD have the following characteristics:

- VPD is available at the rack, drawer, and field replaceable unit (FRU) level.
- For compatibility verification and testing, pluggable FRUs or potentially pluggable FRUs must to be known to the system.
- Uniquely identifies each system hardware, software, and microcode element.
- Becomes part of the VPD record during installation or upgrade.
- When elements do not support VPD in directly readable form, it can be entered manually. Data entered manually is flagged by the operating system software.
- Accessed locally or from a remote console by way of a configuration and VPD facility provided by software.

Customer and Service Personnel Assistance

When field upgrades are made to a system, for example, adding a disk drive drawer to a rack system, the user or service personnel must enter information regarding its physical location and properties using the System Management Interface Tools (SMIT).

VPD Structural Overview

A system-level file or data set contains the fully expanded information on all VPD elements for each enclosure component. The tree structure so formed, shown in Figure 90, begins with a rack or an enclosure level and goes on to identify all system components logically connected.

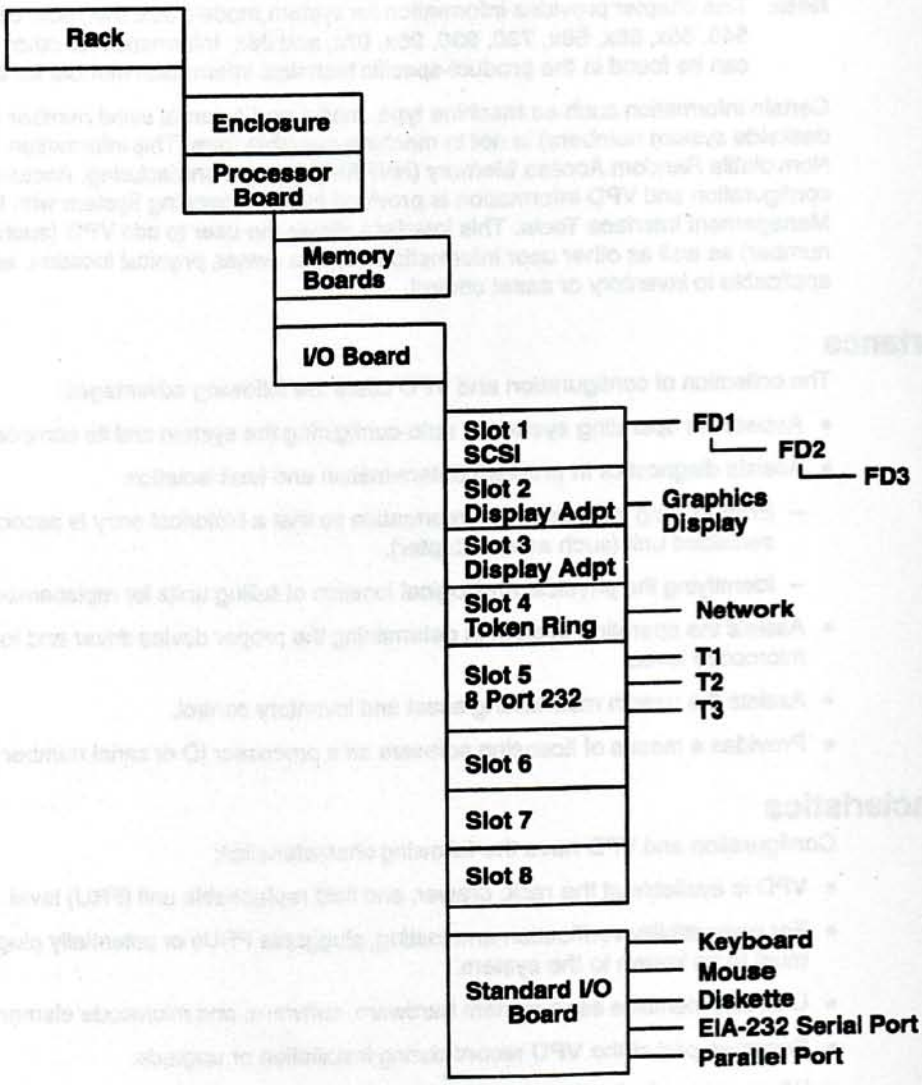


Figure 90. Configuration Tree

System Data Set

The format of the data representing the configuration tree described previously is defined by software. The preferred hardware implementation of vital product data is in the form of keyword descriptors. The VPD is gathered by a software device driver that interfaces with the hardware. If the VPD is stored in a format other than the preferred method, the individual device driver must convert that data into the keyword descriptor format and store that data in a format required by the system configuration and management software method.

Keyword Descriptor Summary

Each keyword header is composed of four bytes of information. The first character is the * (asterisk) character in ASCII format. The next two characters are an abbreviated mnemonic associated with a specific descriptor. The last byte is binary and represents the total length of the keyword descriptor including its header. The length is the total byte count divided by two. Hence, descriptor data is always an even number of bytes with padding as defined by each keyword.

The descriptors listed are a combination of all descriptor keywords used throughout the system. Certain specific types of adapters require pointer values based on the method of implementing VPD.

If a descriptor is manually entered, it must be extended to its full size by the configuration and VPD utility. In addition, the characters ME (for manual entry) are inserted in the high-order positions, adding two characters to its length.

The following list identifies the descriptor keywords currently defined:

- *AD L = addressing field

  The addressing field format is unique to each component described. It must include the Bus Unit ID and slot designation if appropriate. In addition, it specifies sufficient addressing information to program the adapter. The format of the addressing field is specified by software. This descriptor is not present within the machine-readable VPD field contained within an adapter or channel. It is added by software to the configuration and the VPD file or the NVRAM area for VPD.

- *AT L = adapter type

  To support different system field-replacement strategies, this keyword defines a category of Micro Channel adapters. Used in conjunction with the part number (defined by the *PN L and *EC L keywords), this keyword defines a FRU. Its use is not currently planned for the system.

- *CD L = board ID (adapter board ID)

  The board ID field is supplied by software after reading the board ID from POS 0 and POS 1 registers. (Programmable Option Select (POS), replaces switches on feature boards. It is defined under "Micro Channel Adapter Requirements" on page 3-16.) This descriptor only applies to Micro Channel adapters. This descriptor is not present within the machine-readable VPD field contained within an adapter or channel. It is added by software to the configuration and VPD file or the NVRAM area for VPD.

  Following the two bytes of the board ID is a field generated and used by software, which contains mask bytes and POS data used to initialize the adapter. It also contains a flag byte to indicate whether this adapter was successfully configured. The detailed specification of this field is defined by the software operating system.

This descriptor is not present within the machine-readable VPD field contained within an adapter or channel. It is added by software to the Configuration and VPD file or the NVRAM area for VPD.

- *CN L = customer number

  The data portion of this descriptor is in ASCII format. It represents the customer number assigned to the customer owning this machine. The source of this data is the administrative ordering system.

- *DC L = action code and date

  This field is a combined action code and date and time stamp when the action took place. The format of the field is shown in Figure 91:

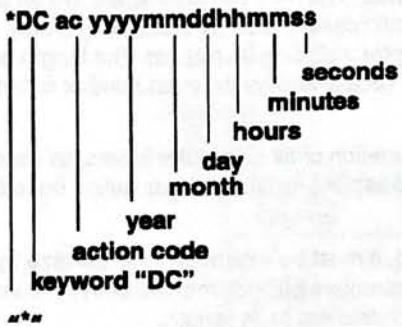


Figure 91. Format of a Combined Action Code and Date and Time Stamp

The following action codes are currently supported:

	Action Code	Description
Plant	BD	Build date (manufacturer)
Plant	AM	Added as MES (MES diskettes)
Plant	AB	Added as BULK MES (Bulk MES diskettes)
Plant	AI	Available at install (manufacturer and field)
Field	ID	Install date (field)
Field	AC	Added with ED (field)
Field	AU	Added from unknown source (field)
Field	AR	Added in repair action (field and CIT)
Field	AT	Added temporarily (field)
Field	AH	Added manually (field)
Field	RU	Removed unknown (field)
Field	RR	Removed in repair action (field and CIT)
Field	RC	Removed with EC (field)
Field	RT	Removed temporarily or powered off (field)
Field	RM	Removed permanently (field)
Field	RN	Removed to another system (field)

- *DD L = device driver level (minimum required)

  The data portion of this descriptor is in ASCII format. It represents the minimum device driver level required. The first release is level 00. Levels are incremented by one for each successive level independently of operating system version and of modification level. The minimum value for L is 3, which is two bytes or two ASCII character numbers of descriptor data plus the header.

  The device driver level represents a generic interface level to software. If the interface changes between software and hardware such that a new interface is required by hardware, the value of this level is incremented. This level is independent of the operating system being used.

  If this keyword is not explicitly specified, level 00 is implied.

- *DG L = diagnostic level (minimum required)

  The data portion of this descriptor is in ASCII format. It represents the minimum diagnostic level required. The first release is level 00. Levels are incremented by one for each successive level independently of operating system version and modification level. The minimum value for L is 3, which is two bytes or two ASCII character numbers of descriptor data plus header.

  The diagnostic level represents a generic interface level to diagnostics. If the interface changes between software and hardware such that a new interface is required by hardware, the value of this level is incremented. This level is independent of the operating system being used.

  If this keyword is not explicitly specified, level 00 is implied.

- *DL L = drawer level

  The data portion of this descriptor is in ASCII format and specifies a drawer location in Electronics Industries of America (EIA) units. It represents the drawer location within a rack for an enclosure. The EIA unit values are marked on the rear panel of the rack. These values are captured during manufacturing while a rack is in its final manufacturing test. In the field, configuration changes that alter drawer information must be supplied by the trained customer or customer engineer installing the change.

- *DS L = displayable message (ASCII format)

  This is an optional field that can include a message to be printed or displayed for this record type. Avoid the ASCII character * (asterisk) within the data content of this message.

  Micro Channel adapters designed for the system unit require this keyword with a brief description of the adapter function.

- *DU L = drawer unit

  This field is used at the system level to describe the contents of a drawer unit within a rack system. The number in this field can be a feature code, a machine type and model number, or other alphanumeric field used to describe the drawer unit. The data portion is in ASCII format.

- *EA L = electronic address

  The data portion of this descriptor is in ASCII format. The value represents an electronic address where this machine can be contacted. This field must be entered manually by the "Product Topology Service Aid."

- *EC L = engineering change level

  The data portion of this descriptor is in ASCII format. The characters are alphanumeric and represent the engineering change level for this element. The values of L, which range from 6 to 8, represent descriptor data counts of 8 to 12 alphanumeric characters. This descriptor number is left justified and can be padded with low-order blanks. For IBM released parts, this field must contain the IBM EC number.

- *FC L = feature code

  This field contains the feature code or RPQ number used to order or specify the hardware described after it in the product topology data hierarchy. The designation must match precisely the nomenclature used by the order process for the device. The source of this data is the administrative order entry system.

- *FN L = FRU number

  The data portion of this descriptor is in ASCII format. The characters are alphanumeric and represent the assigned Field Replaceable Unit part number for this element of the system product. The value of L ranges from 6 to 8 representing descriptor data counts from 8 to 12 alphanumeric characters. The data is right justified and padded with high-order zero. For IBM released parts, this field must contain the IBM FRU Part Number.

- *LA L = pointer to loadable microcode on the adapter

  This keyword is an optional descriptor type available for use. If an adapter chooses to implement loadable microcode using the POS registers for writing and reading of microcode, this field is required. Micro Channel adapters can use the POS subaddress facility or any other method to implement loadable microcode. Data in the field can be encoded in binary on the device but is externalized in ASCII or a hexadecimal representation of a binary value in ASCII.

  The data portion of this descriptor is an address pointer in the POS subaddress space. Byte 0 is the most significant address byte, and byte 1 is the least significant address byte in binary.

- *LL L = loadable microcode level (minimum required)

  The data portion of this descriptor is in ASCII format. It represents the minimum loadable microcode level required for functional operation. The first release is level 00. Levels are incremented by one for each successive level. Loadable microcode is associated with a given board ID rather than a part number or EC level. Therefore, as changes are made to a particular adapter, a corresponding microcode level can be required for correct operation. This field is required if loadable microcode is required for functional operation of the adapter. The field's presence notifies the initialization code of this additional requirement. The minimum value for L is 3, which is two bytes or two ASCII character numbers of descriptor data plus the header.

  This is a generic level equivalent in use to a device driver or a diagnostic level. It indicates that a significant change was implemented on the adapter and that a new minimum level of loadable microcode is required.

- *LO L = location (internal or external)

  This descriptor is optional. The data portion of this optional descriptor contains the ASCII characters IN for internal devices or EX for external devices or for other components. The default value for this descriptor is EX and is implied if this field is not specified. This field is generated dynamically by software for fixed disks attached to a SCSI adapter that provides internal reset capability. For other devices, it can be entered by the user in the

configuration and VPD utility. It is required for power domain and security domain requirements. The value of L is 3.

- *MF L = manufacturer

  The manufacturer descriptor field is typically six characters of ASCII data. For our components, the first three characters are alpha characters. The next three characters are alphanumeric and are a code assigned to each location. For six characters of descriptor data, L equals 5.

  Vendor manufacturers are identified by a 6-digit number assigned by the purchasing department when a contract is established. An abbreviation for the location establishing the contract is concatenated to the purchase order number.

  The *MF L keyword is being retired and replaced with the *MN Keyword.

- *MN L = Manufacturer and location

  The manufacturer descriptor field is 4 or 10 characters of ASCII data.

  - For an IBM manufactured component (built for IBM), the first character is an ASCII number "1" character. The next 3 characters are assigned by IBM and are a location code (LOC) assigned to each IBM location (described in the following list).
  - For an IBM manufactured component (built for an OEM), the first character is an ASCII number "2" character. The next 3 characters are assigned by IBM and are a location code (LOC) assigned to each IBM location (described in the following list).
  - For a Vendor manufactured component (built for IBM) The first character is an ASCII number "3" character. The next 3 characters are assigned by IBM and are a location code (LOC) assigned to each IBM location (described in the following list). This is followed by a 6–digit number (NNNNNN) assigned by the IBM purchasing department when a contract is established.
  - For an OEM manufactured component. The first character is an ASCII number 4 character. Up to 9 additional characters may be assigned as a manufacturer identification. These 9 additional characters are assigned by the OEM.

Location Code	Manufacturing Location
966	Austin, ESD plant
97N	Austin, ESD card manufacturer
975	Boca Raton, ESD plant
9NX	Boca Raton, Card vendor
98J	Boulder, IPD plant
984	Burlington, GTD plant
983	Charlotte, Card manufacturer
955	Endicott, CP manufacturer
991	Endicott, SP manufacturer
981	Lexington, IPD plant
997	Manassas, GTD
988	Raleigh, CPD plant
988	Rochester, SPD plant
98K	Tucson, CP plant
90S	Bromont, plant
90W	Greenock, plant
90F	Toronto, plant
90Q	Vimercate, plant

- *NA L = network address

  This is an optional field used by those adapters which require a unique network address for a local area network. Adapters such as token ring and baseband use this field. Data in the field can be encoded in binary on the device but is externalized in ASCII or a hexadecimal representation of a binary value in ASCII.

  When specified, this field must be implemented as the first descriptor keyword and therefore "NA L" is located at address 00 08. The first data byte is, therefore, located at byte 12 (decimal) or 00 0C Hex within the extended storage area located by POS Registers 6 and 7.

- *NX L = pointer to next adapter VPD for multiboard adapters

  This is used by multicard adapters including those occupying more than one card slot. The primary card must provide POS registers. Additional (secondary) cards must be plugged into slots adjacent to the primary card. This field specifies the VPD address to be specified in POS registers 7 and 6, respectively, in order to access VPD data on the adjacent (secondary) adapters. Data in the field can be encoded in binary on the device but is externalized in ASCII or a hexadecimal representation of a binary value in ASCII.

- *OS L = Operating System level

  The data portion of this descriptor contains the name of the operating system (for example, "AIX") followed by version, modification, and PTF level. All characters are specified in ASCII. Additional data can be included to specify specific options being used (such as cluster). This descriptor is required in the Enclosure Record store in NVRAM and in the configuration and VPD file.

- *PC L = processor component definition

  This data represents binary information that details the processor speed and model.

- *PI L = processor ID

  The data portion of this descriptor is an ASCII alphanumeric field that represents the processor ID for a processor enclosure. This data is normally extracted from IPL ROM associated with the processor board. This serial number is often used for software licensing.

- *PN L = part number

  The data portion of this descriptor is in ASCII format. The characters are alphanumeric and represent the part number for this element. The values of L, which range from 6 to 8, represent descriptor data counts of 8 to 12 alphanumeric characters. This descriptor number is right justified and can be padded with high-order zeros. For IBM released parts, this field must contain the IBM Part Number.

- *RA L = pointer to ROM code on adapter

  If an adapter chooses to access on-board ROM using the POS registers for reading microcode, then this field is used. Data in the field can be encoded on the device in binary, but is externalized in ASCII or a hexadecimal representation of a binary value in ASCII. The first data byte represents a POS register to use as a Port to read and write data to the adapter for purposes of reading microcode on the adapter. Any POS register (0–5) can be specified. The second byte specifies the number of low-order bit positions of POS register 5 to use for expanding the address range of POS registers 6 and 7. The address so formed is specified as follows:

  ```
  POS 5 (n low-order bits), Pos 7, POS 6
  ```

The second byte can specify from 0 to 6 bits of additional addressability. Data bytes 3, 4, 5, and 6 specify the initial address for reading microcode. This is an optional descriptor type available for use.

- *RL L = ROM level and ID

  This descriptor identifies the part number of any nonalterable ROM code on the adapter. The data field of the keyword is defined as follows:

  Bytes 0–11 Part number of the ROM code (alphanumeric ASCII).

  Bytes 12–23 EC level of ROM code (alphanumeric ASCII), this is optional if the ROM code PN is not changed when updated.

- *RM L D = Alterable ROM ID

  This descriptor identifies the part number of any alterable ROM code on the adapter. The data field of the keyword is defined as follows:

  Byte 0 An optional "field patch level." A value of 0 indicates no field patch applied (ASCII).

  Bytes 1–12 Part number of the ROM code (alphanumeric ASCII).

  Bytes 13–24 EC level of ROM code (alphanumeric ASCII), this is optional if the ROM code PN is not changed when updated.

- *RN L = rack name (letter designation)

  This keyword is a required descriptor for records describing a rack enclosure. The abbreviated name consists of a 2 ASCII character field, such as "space A" or "space B," that matches the letter installed on the rear of the rack unit. It is used by diagnostics for FRU location specification.

- *RW L = pointer to Read and Write adapter registers

  This keyword is an optional descriptor type available for use. If an adapter chooses to implement Read and Write registers using POS registers, then this field is used. Adapters can use the POS extended addressing facility or any other method to implement access to Read and Write registers and storage.

  Data in the field can be encoded in binary on the device byte is externalized in ASCII or a hexadecimal representation of a binary value in ASCII. This first data byte represents a POS register to use as a port to read and write data to the adapter for specific adapter purposes. Any POS register (0–5) can be specified. The second byte specifies the number of low-order bit positions of POS register 5 to use for expanding the address range of POS registers 6 and 7. The address so formed is specified as follows:

  ```
  POS 5 (n low-order bits), Pos 7, POS 6
  ```

  The second byte can specify from 0 to 6 bits of additional addressability. Data bytes 3, 4, 5, and 6 specify the initial address for accessing Read and Write registers or storage. The size and use of this Read and Write area is adapter specific. The minimum value for L is 5, which represents 6 bytes of descriptor data plus a keyword.

- *SC L = specify codes

  This field contains all the specify codes selected for this machine. The source of this data is the administrative order entry system.

- *SE L = machine serial number

  This field contains the serial number assigned to the processor machine type by the manufacturing location. The number normally begins with two digits which uniquely identify the plant of manufacture. These are followed by a 5 character serial number. For example, in the serial number 2605668, "26" is the Austin plant designation and "05668" represents the serial number of this machine. The source of this data is the administrative order entry system.

- *SL L = slot location

  Memory board adapters use this description to specify board slot location. The data field is 2 bytes in size.

- *SN L = serial number

  The data portion of this descriptor is in ASCII format. The characters are alphanumeric and represent the serial number of the machine or device. The value of L is 6, representing a descriptor data count of 8. The descriptor number is left justified and can be padded with low-order blanks.

- *SY L = system number

  This field contains the system number assigned to this system. The source of this data is the administrative order entry system.

- *SZ L = size

  Memory board adapters use this description to specify the size in M bytes. The data portion contains 1 to 8 digits, left-justified, with no leading zeros and padded on the right with blanks as required.

- *TM L = machine type and model

  The data portion of this descriptor specifies the machine type in ASCII format. The data portion is 4 characters long, followed by a dash (–) and the 3 character machine model. The total data length is 8 characters. Therefore, L is specified as 6, representing 8 characters of data plus the header (for example, '7207–001').

- *US L = user data

  The data portion of this field is an ASCII character string specified by the user utilizing the configuration and VPD utility. It could be used to specify owner, location, or similar information. It must contain an even number of bytes.

- *VE L = pointer to VPD extended data on adapter

  This optional descriptor is used as an address pointer in the subaddress space of VPD for a Micro Channel adapter. It points to a storage location that contains additional keyword descriptors in order to support an implementation of noncontiguous keyword descriptor data.

  The data portion of this descriptor is an address pointer in the POS subaddress space. Byte 0 is the most significant address byte, and byte 1 is the least significant address byte in binary form.

- *Z0–*Z9, ZA–ZZ L = available for adapter-specific use.

  Refer to the specific adapter section for a description.

Hardware VPD Descriptor Summary

The following sections define the minimum requirements of various hardware components of a system.

Rack Record

Descriptors

The required descriptors for the rack record are as follows:

Keyword	Description
*PN L	Part number
*EC L	Engineering change level
*FN L	Field replacement unit number
*TM L	Machine type and model (for the primary rack)
*FC L	Feature code (for secondary, attached racks)
*SN L	Serial number
*MF L	Manufacturer
*RN L	Rack name (letter designation).

Implementation Notes

Rack configuration data is supplied by manufacturing in NVRAM. The rack name is a letter designation (A, B, C) used by diagnostic programs to locate problem FRUs. This information must be input by a customer engineer from the hard card using a configuration and system management utility if this unit is field installed. The serial number specified must match the external label on the system unit.

Enclosure Record

Descriptors

The required descriptors for the enclosure record are as follows:

Keyword	Description
*PN L	Part number
*EC L	Engineering change level
*FN L	Field replacement unit number
*SN L	Serial number (externally visible)
*TM L	Machine type and model
*DL L	Drawer level (if rack-mounted)
*MF L	Manufacturer.

Implementation Notes

An enclosure represents a physical package. It can be a drawer in a rack, a deskside system, a table-top system, a portable file, a free-standing tape drive, or other free-standing unit. Enclosures are normally machine type and models; however, feature codes can also be designated.

This information must be input by a customer engineer from the hard card using a configuration and system management utility if this unit is field-installed. The serial number specified must match the external label.

Processor Board Record

Descriptors

The required descriptors for the processor board record are as follows:

Keyword	Description
*PN L	Board part number
*EC L	Engineering change level
*PI L	Processor ID
*FN L	Field replacement unit number
*RL L	ROM level and ID (IPL ROM)
*RL L	ROM level and ID (on card sequencer (OCS) ROM)
*RL L	ROM level and ID (seeds ROM)
*PC L	Processor component definition (specifies speed and processor model)
*Z0 L–*Z9 L	Processor module information.

Implementation Notes

The board description represents a reflection of the physical packaging of a processor unit. The processor board is the physical unit that contains the processor modules.

I/O Board Records

Descriptors

The required descriptor for the I/O Board record is as follows:

Keyword	Description
*EC L	Engineering change level.

Implementation Notes

The I/O Board contains the I/O slots for installing I/O adapters. If a model contains only a system board or a combination board, the value in the System I/O register designates the level of the hardware components supporting the interface to the logic normally associated with the I/O Board.

As currently implemented in most models, the I/O Board level is identified by an 8-bit code in a System I/O register. Each level is incremented by one. Software locates the corresponding part number and the EC level by table lookup.

Memory Records

Descriptors

The required descriptors for the memory records are as follows:

Keyword	Description
*PN L	Part number
*SN L	Serial number
*FN L	Field replacement unit number
*MF L	Manufacturer
*SZ L	Size in megabytes
*EC L	Engineering change level
*SL L	Slot location (software)
*Z0 L	EC level Left Data Multiplexer module
*Z1 L	EC level Right Data data multiplexer module
*Z2 L	EC level Controller module
*Z3 L	SIMM product definition (PD) code.

Implementation Notes

The initial memory board does not support VPD. The default data of all zeros is written to the board immediately after startup. If the board is revision level 2 or higher, the real VPD is returned on the first read operation. If the board is revision level 1 (initial release), all zeros are returned on the first read operation.

Extra I/O Board Record

The keywords specified depend on the function provided by the board. The function should be compatible with the requirements for a system, an I/O Board, or other adapters. The minimum requirements always include the *PN and *EC keywords.

SCSI Attached Device Records

The exact information can vary from vendor to vendor; however, the data supplied by the **Inquiry** command on the SCSI interface contains machine type and model, part number, EC or revision level, serial number, and microcode information (the RL and LL keywords as appropriate). Some units provide VPD for the device enclosure unit as well as data for the logic board associated with the unit, where each can be a FRU. Serialization is *always* required. Software must provide a FRU number if one is not contained in the machine-readable VPD.

Device Required Descriptors

The required device descriptors are as follows:

Keyword	Description
*PN L	Part number
*EC L	Engineering change level
*FN L	Field replacement unit number
*TM L	Machine type and model
*SN L	Serial number (matches external bar code label)
*MF L	Manufacturer.

Optional Descriptors

The optional device descriptors are as follows:

Keyword	Description
*RL L	ROM level and ID (if ROM is present)
*LL L	Loadable ROM level and ID (minimum level required).

Standard I/O Attached Devices

The exact data can vary from device to device making the ROM level (RL) and loadable microcode level (LL) conditionally required.

Device Required Descriptors

The required device descriptors are as follows:

Keyword	Description
*PN L	Part number
*EC L	Engineering change level
*TM L	Machine type and model
*FN L	Field replacement unit number
*SN L	Serial number (matches external bar code label)
*MF L	Manufacturer.

Conditionally Required Optional Descriptors

The optional device descriptors are as follows:

Keyword	Description
*RL L	ROM level and ID (if ROM is present)
*LL L	Loadable ROM level and ID (minimum level required).

Micro Channel Adapter Requirements

The preferred method of implementation is to use the Programmable Option Select (POS) register subaddressing space during board setup. When POS registers 6 and 7 contain values other than X'0000', POS register 3 is a *port* that accesses a read-only memory (ROM or EPROM) module containing vital product data in the keyword descriptor format. For example, when POS register 6 equals X'01' and POS register 7 equals X'00', a one-byte load operation from POS register 3 reads data from address X'0001' in the EPROM containing VPD. When POS register 6 equals X'02' and a load from POS register 3 of 1 byte reads from the address X'0002', and so forth. An alternative address is X'FF01'.

A header is defined that immediately precedes memory containing the descriptor keywords. It is recommended that a pluggable EPROM be written at the time of manufacture on a part-by-part basis (for serialization and incorporation of the latest EC level information).

An alternative method of machine-readable vital product data (VPD) allows the adapter to provide the data in an adapter-specific manner. For example, available ROM locations could be used in a fixed-memory location known to the device driver for this adapter. The device driver must gather and convert the VPD into the keyword format described for the preferred method. The device driver then provides the information to the operating system in the manner required by the individual operating system. This alternative method allows existing adapters to add VPD with the least hardware impact.

Most adapters designed for the system have implemented the preferred method with the required keywords defined in the following lists:

- Required keywords:

Keyword	Description
*PN L	Part number
*EC L	EC level
*FN L	FRU number for field replacement unit
*SN L	Serial number
*MF L	Manufacturer and location.

- Conditionally required keywords:

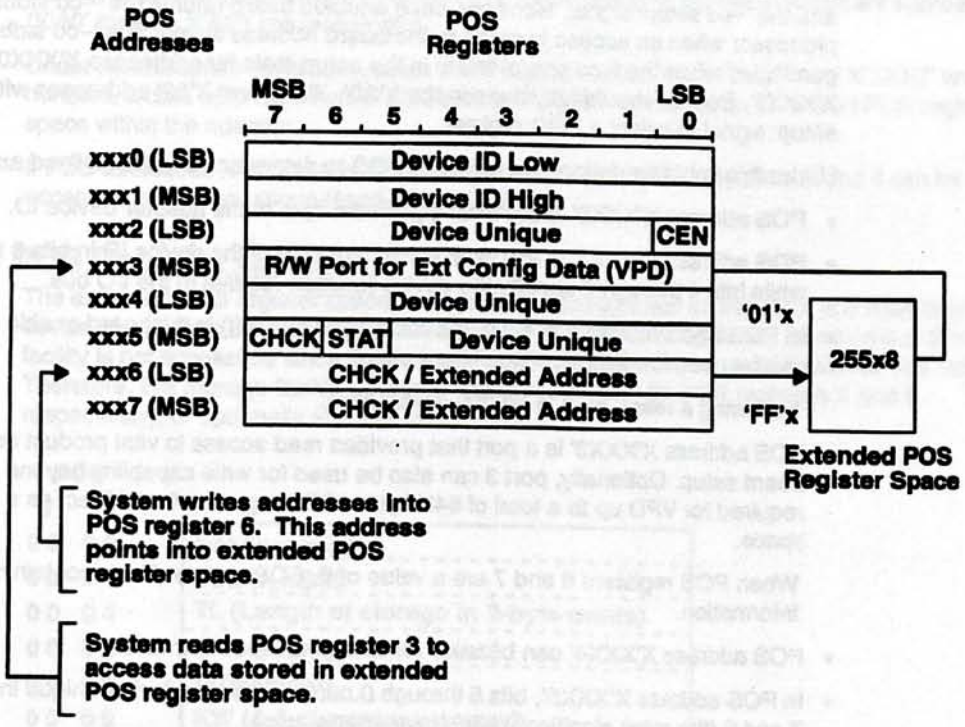
Keyword	Description
*DS L	Brief description (for example, SCSI, token ring, and 8-port asynchronous adapter)
*RL L	ROM level and ID information (if ROM is present)
*LL L	Loadable microcode level (if loadable code is present)
*NA L	Network address (if adapter type requires a network address)
*DD L	Device driver level
*DG L	Diagnostic level.

- Optional keywords:

Keyword	Description
*RA L	Pointer to ROM code on adapter
*RW L	Pointer to Read and Write Adapter registers
*DS L	Displayable message (additional description)
*LA L	Pointer to loadable ROM code on adapter
*Z0 L – *ZZ L	Available for adapter-specific use.

Preferred Implementation – POS Configuration Registers

The POS addresses for the POS registers are shown in Figure 92.



Note: POS register 6 is initialized to a value of 0 when the power is turned on. A nonzero value must be written to POS register 6 to access the extended POS register space.

Figure 92. POS Configuration Registers

Term	Description
MSB	Most significant byte
LSB	Least significant byte.

TL The total length in 2-byte words to read from this facility beginning at address X'00 08' to the end of the last data field. This field is two bytes in binary format.

CRC Value This 2-byte value is a cyclic redundancy check (CRC) value starting at address X'00 08' through the end of storage (TL).

The CRC polynomial is 1 + X (exp 5) + X (exp 12) + X (exp 16), which is the same as the CRC polynomial used for most diskette records.

Sample Layout of the Micro Channel Adapter VPD

Address (Hex)	Contents of ROM and PROM (ASCII numbers in parentheses are decimal, 1-byte values)
00 01	V P D (00) (40) (252) (188)
00 08	* P N (06) 6 1 8 1 6 8 2 A
00 14	* E C (07) 4 9 5 0 2 6 2 5 3 6
00 22	* S N (06) 0 0 0 0 0 1 9 4
00 2E	* F N (05) 1 3 5 7 2 2
00 38	* M F (05) I B M 0 3 7
00 42	* D S (05) 8 – P O R T
00 4C	* D G (03) 0 1
00 52	* D D (03) 0 1
00 58	– – – – – – – – – – – – – – – –

Notes:

1. The CRC value on data from X'00 08' through X'00 57' is the actual calculated CRC for this example data.
2. A – (dash) indicates binary zeros.
3. A ( ) (parenthesis) indicates decimal byte length divided by 2.

Address (Hex)	Contents of ROM and PROM (Hex)
00 01	56 50 44 00 28 FC BC
00 08	2A 50 4E 06 36 31 38 31 36 38 32 41
00 14	2A 45 43 07 34 39 35 30 32 36 32 35 33 36
00 22	2A 53 4E 06 30 30 30 30 30 31 39 34
00 2E	2A 46 4E 05 31 33 35 37 32 32
00 38	2A 4D 46 05 49 42 4D 30 33 37
00 42	2A 44 53 05 38 2D 50 4F 52 54
00 4C	2A 44 47 03 30 31
00 52	2A 44 47 03 30 31
00 58	– – – – – – – – – – – – – – – – – – – –

Chapter 4. Initial Program Load (IPL) ROM

Chapter Contents

Description

The initial program load (IPL) is the sequence of events that occurs during the period of time following a power-on reset or system reset operation until control of the processor is passed to loaded code.

The IPL consists of initializing and testing the base hardware, and then finding, loading, and executing code. The task of the read-only memory (ROM) resident IPL function is to verify the portion of the machine necessary to initialize the IPL function, and then to start the IPL if possible.

ROM Hardware

- ROM is located on the processor board.
- ROM addressing begins at X'FFF00000'.
- IPL ROM code entry point address is X'FFF00100'.
- The configuration information is contained in ROM. The following configuration information is required:
  - Processor board engineering change (EC) level and part number
  - Processor serial number
  - ROM part number and ID
  - ROM copyright
  - ROM version and level.

Hardware Initialization

Prior to execution of IPL ROM code, hardware initialization puts the processor into a known working state.

For system units with the on card sequencer (OCS), hardware initialization is performed by the OCS before control is passed to IPL ROM code.

Cold System Reset

Cold system reset occurs at initial startup and in system units with on card sequencer (OCS) when a hardware event (such as check stop) triggers the system reset finite state machine and the resulting system reset count is not equal to 0. Following hardware initialization by OCS, a System Reset interrupt occurs at X'FFF00100' in IPL ROM.

Warm System Reset

A warm system reset occurs when a hardware event triggers the system reset finite state machine and the resulting system reset count is equal to 1. A System Reset interrupt occurs and normally (machine state register (MSR) IP bit equals 0) execution proceeds at location X'00000100' in the operating system. The operating system can perform actions such as dumping all or part of memory or invoking a debugger and then can reload the operating system kernel. (If the MSR IP bit equals 1, execution proceeds at X'FFF00100', and a cold IPL occurs.)

ROM Warm IPL Function

An entry point is provided in IPL ROM to facilitate reloading of the code specified in the IPL record. The ROM warm IPL function reloads the IPL record and code specified in the IPL record and passes control to the code while disturbing the existing machine state as little as possible. The hardware is not reinitialized. The IPL device is redetermined.

Note: Upon receipt of a warm system reset interrupt, an operating system can elect to reload itself without branching to ROM.

Hardware-Initiated IPL

The following events cause hardware to generate a System Reset Interrupt:

- Power-on reset (POR).
- Reset button on operator panel pushed. Keyswitch lock enables the Reset button.
- Check stop for system units with OCS.

Software-Initiated IPL

A ROM warm IPL can be achieved by branching to the warm IPL entry point in ROM.

Software can designate the IPL device by way of the device lists in nonvolatile random access memory (NVRAM). Software can expedite the IPL process by designating a known IPL device near the front of the device lists. Only devices for which there is an IPL control block entry indicating the device is present and functional are eligible as IPL devices. Software must provide a method for the operator to customize the device lists in NVRAM. If the operator elects not to specify a device list, the ROM uses a predefined default list.

No special entry point has been defined in the IPL ROM to facilitate a software-initiated cold IPL.

Check Stop

For system units without OCS, (a check stop event causes a halt) the check stop count in NVRAM is always a value of 0.

For system units with OCS, a check stop event causes a cold system reset.

Before executing the power-on self test (POST), the IPL ROM inspects the check stop count in NVRAM:

- A value of 0 indicates that a check stop event did not occur. The IPL ROM continues normal execution.
- A value of 1 indicates that a check stop event occurred and that OCS logged out check stop data in NVRAM. The IPL ROM continues normal execution.
- A value greater than 1 indicates that an error occurred, which caused a check stop event. The error was not detected by the OCS built-in self test (BIST). The IPL ROM puts an error code in the light-emitting diodes (LEDs) and halts.

LEDs

The system units have three 7-segment LEDs on the operator panel. The IPL ROM displays appropriate values in the LEDs to indicate the progress of the IPL and to identify the point of the error should a fatal error occur.

NVRAM

The system units have at least 8K bytes of NVRAM.

If NVRAM is valid, the IPL ROM reads the following information from NVRAM:

- IPL expansion code
- Normal device list
- Service device list
- Network boot information.

IPL Record

In order to perform an IPL, a valid IPL record must reside on a valid IPL media. This record consists of the following:

- An ID uniquely identifying it as an IPL record.
- A media description, such as characteristics of the IPL device.
- One or more load descriptions, such as location, length, and entry point of code to be loaded (service or normal).
- The address where the code must load.

The IPL record format is common for all devices.

Security

A Keylock switch in the secure position disables the Reset button on the operator panel. In the normal position, the Keylock switch permits the IPL to initialize only from trusted IPL devices. In the service position, the Keylock switch allows the IPL to initialize from any IPL device.

The following are characteristics of the IPL device:

- Disabling of Reset button is a hardware function. Disabling stops the machine from performing an IPL.
- Disabling of the IPL from devices other than trusted IPL devices is implemented in the IPL ROM. The IPL ROM controller code senses the position of the keyswitch and if in the normal position, only permits an IPL from trusted IPL devices. If a valid IPL record and IPL code are found on a trusted IPL device, the IPL sequence completes; otherwise, the IPL ROM loops, polling the trusted IPL devices for an IPL record and testing for a change in keyswitch position.

Service IPL

The IPL ROM supports an IPL from an alternate load description. For systems with a service keyswitch position, when the keyswitch is in the service position, the IPL ROM ignores the primary (normal) load description in an IPL record and loads the software described by the alternate (service) load description. The IPL ROM inspects the code length fields in the primary and alternate load descriptions to determine what can be loaded from a particular device. The length field must be a value of 0 if the code is not present.

This function is provided so that diagnostics or another alternate operating environment can initialize the IPL from the same device as the operating system.

IPL ROM Components

The IPL ROM code is functionally divided into the power-on self tests, the device interface routines, and three control programs:

- Initial sequence controller (ISC)
- Core sequence controller (CSC)
- IPL controller (IPLC).

Initial Sequence Controller

The initial sequence controller (ISC) accepts control after hardware initialization and passes control to the Core sequence controller (CSC) after completion. The following diagram gives a general idea of what the ISC does.

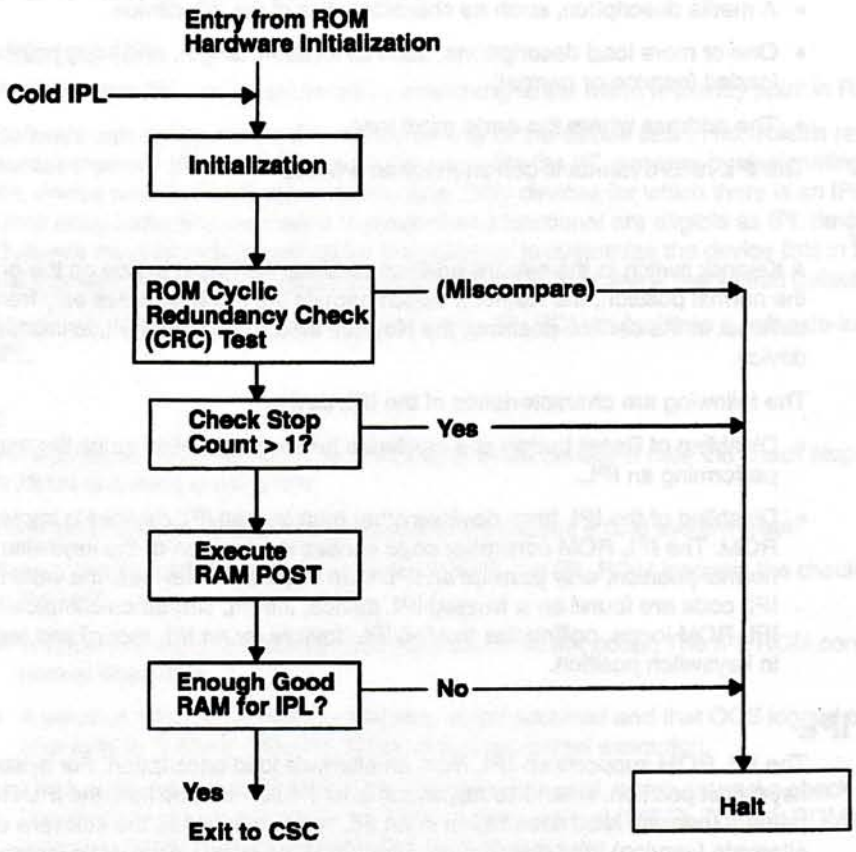


Figure 94. Initial Sequence Controller Logic Flow

The following are major initial sequence controller functions:

- Performing initialization
  - Reading ROM configuration information from non-CRC checked part of ROM and set ROM size and speed in the Storage Control Unit Configuration register (SCCR)
  - Setting initial LED values
  - Performing other initialization as required.
- Activating system ROM cyclic redundancy check
  - Halting if miscompare.
- Inspecting check stop count
  - If 0 or 1, continuing normal execution
  - If greater than 1, halting with an error code in LEDs.
- Executing RAM POST
  - Determining memory configuration (includes setting configuration register extents).
  - Finding enough good memory. At least 1M-byte memory is required. (2M-bytes memory are required on some systems.)
  - Testing memory and creating a bit map.
  - Storing results of RAM POST into the IPL control block.
- Inspecting return code from the RAM POST.
  - Halting if the amount of good memory is less than required for the system.

Core Sequence Controller

The core sequence controller accepts control from the initial sequence controller and passes control to the IPL controller. The core sequence controller sequences through the POSTs. These POSTs complete the testing performed by the IPL ROM. The following diagram gives a general idea of what the core sequence controller does.

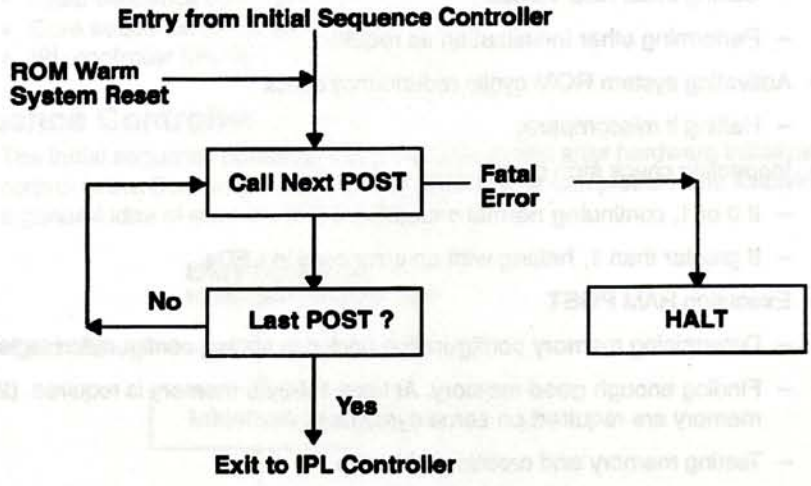


Figure 95. Core Sequence Controller

The following are functions of the core sequence controller:

- Executing POSTs in a predefined order
- Passing a pointer from POSTs to the IPL control block to record results
- Passing return codes from POSTs to the Core sequence controller.

IPL Controller

The IPL controller accepts control from the core sequence controller and passes control to loaded code. The following diagram gives a general idea of what the IPL controller does. It is the job of the IPL controller to find a successful IPL path. If an IPL attempt is not successful, the IPL controller continues to cycle through the IPL device list (DevList), trying to initiate an IPL from each IPL device.

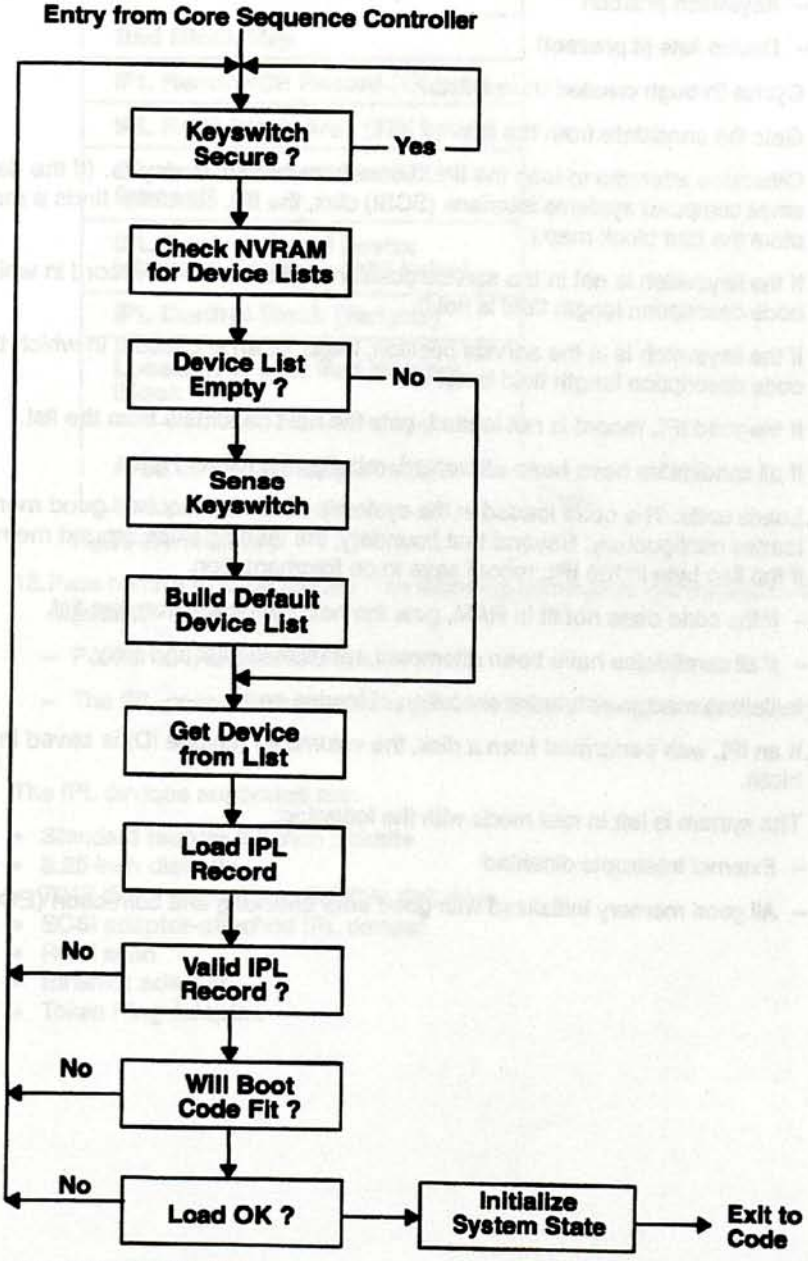


Figure 96. IPL controller

IPL Controller Functions

The following are functions of the IPL controller:

1. NVRAM CRC test. Run NVRAM cyclic redundancy check on portions of NVRAM containing configured IPL device selection sequence.
2. Builds the list of IPL device candidates based on the following:
   - Keyswitch position
   - Device lists (if present).
3. Cycles through created device lists.
4. Gets the candidate from the list.
5. Otherwise attempts to load the IPL record from candidate device. (If the device is the small computer systems interface (SCSI) disk, the IPL controller finds a memory area to store the bad block map.)
6. If the keyswitch is not in the service position, looks for an IPL record in which the primary code description length field is not 0.
7. If the keyswitch is in the service position, looks for an IPL record in which the alternate code description length field is not 0.
8. If the valid IPL record is not loaded, gets the next candidate from the list.
9. If all candidates have been attempted, rebuilds the list and retry.
10. Loads code. The code loaded in the system's minimum required good memory space is loaded contiguously. Beyond that boundary, the loading skips around memory bad blocks if the flag byte in the IPL record says to do fragmentation.
    - If the code does not fit in RAM, gets the next candidate from the list.
    - If all candidates have been attempted, rebuilds the list and retry.
11. Initializes machine state for execution of loaded code.
12. If an IPL was performed from a disk, the volume ID (unique ID) is saved in the IPL control block.

    The system is left in real mode with the following:
    - External interrupts disabled
    - All good memory initialized with good error checking and correction (ECC).

- Any IPL device used inactive
- Memory contents as shown in Figure 97.

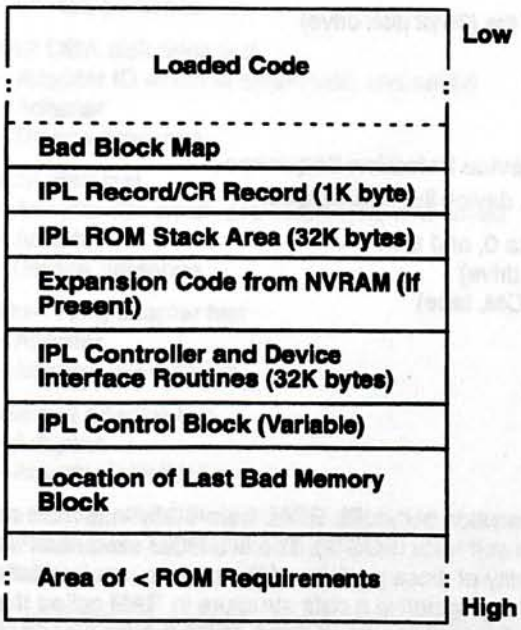


Figure 97. RAM map

13. Pass control to code loaded. The following parameters are passed to the loaded code in registers:
    - Pointer to IPL control block.
    - The IPL control block contains pointers to other things (such as memory bit map).

IPL Devices

The IPL devices supported are:

- Standard feature 3.5-inch diskette
- 5.25-inch diskette
- 7012 direct bus-attached (DBA) disk drive
- SCSI adapter-attached IPL devices
- ROM scan
- Ethernet adapter
- Token Ring adapter.

Trusted (Normal) Default IPL Device Selection Sequence

The following sequence is the trusted (normal) default IPL device order:

1. ROM scan
2. Direct bus-attached file (7012 disk drive)
3. SCSI device
4. Token Ring adapter
5. Ethernet adapter.

Service Default IPL Device Selection Sequence

The default service IPL device list is as follows:

1. Standard I/O diskette 0, and then 1
2. DBA file (7012 disk drive)
3. SCSI device (CD-ROM, tape)
4. ROM scan
5. SCSI device (disk)
6. Token Ring adapter
7. Ethernet adapter.

Power-On Self Tests

Tests run during the execution of the IPL ROM, before any load from an IPL device, are referred to as power-on self tests (POSTs). The IPL ROM executes POSTs to determine the presence and functionality of those portions of the system required for a successful IPL. The results of these tests are collected in a data structure in RAM called the IPL control block. The IPL ROM testing is limited to those portions of the machine necessary for an IPL: the base system (RAM and I/O Channel Controller) and the IPL devices. The IPL ROM code does not halt due to the absence or failure of hardware except where that absence or failure directly precludes the IPL.

If an error is detected during a POST, information about the error is returned for resolution.

Except for base system function, testing performed by IPL ROM POSTs is minimal. The IPL device POSTs test an adapter's functionality and device presence. The following tests are performed:

- RAM POST
- I/O channel controller (IOCC) POST
- IPL device POSTs.

RAM POST

- Processor and memory interface tests (Memory Control Unit)
- Memory test.

IOCC POST

- Processor and IOCC interface tests
- IOCC register tests
- Bus test (IOCC to Standard I/O)
- Direct memory access (DMA) test
- Test interrupts.

IPL device POSTs

- Standard and feature diskette drive test
  - Adapter
  - Device presence.
- 7012 DBA disk drive test
  - Adapter ID which is determined and saved
  - Adapter
  - Device presence.
- SCSI disk test
  - Adapter IDs which are determined and saved
  - Adapter
  - Device presence.
- Token Ring adapter test
  - Adapter
  - Adapter initialization.
- Ethernet adapter test
  - Adapter
  - Adapter initialization.

Before calling a POST routine, the controller puts a value in the LEDs identifying the POST so that if an error occurs while a POST is running and control does not return, the error is identifiable.

POST routines are passed a pointer that identifies to the area of the IPL control block in which to store the test results. See Figure 97 on page 4-11 for more information.

IPL ROM Functional Characteristics

The following section describes the IPL ROM entry points, control block, configuration records, NVRAM, expansion code, and LED operation.

Cold IPL Entry Point

The ROM entry point is at real address X'FFF00100'. This is the normal entry point following power-on reset.

ROM Warm IPL Entry Point

An entry point is provided in IPL ROM to facilitate the reloading of the system after a warm system reset. The entry point results in an IPL record and code being reloaded. On a warm IPL, the system must pass the IPL control block pointer in general purpose register 3. The pointers in the IPL control block are considered valid and reusable.

The ROM warm IPL entry point is stored in the ROM entry point table. A pointer to the ROM entry point table is stored in the IPL control block by the IPL ROM.

The following requirements must be met to perform a ROM warm IPL:

- IPL ROM code operates in real mode.
- ROM is mapped to real address X'FFF00000' at startup.
- The IPL control block must be in memory, and a pointer to it must be passed to ROM in register 3.

- The contents of the IPL control block, as saved by the previous execution of the IPL ROM, must be intact. (The operating system must not delete the existing contents of the IPL control block.)
- The linkage conventions and the register conventions established by the IPL ROM must be followed.
- The IPL ROM code can alter the contents of memory.

IPL Control Block

The IPL control block is created in RAM during the execution of ROM. The IPL control block size is variable. The IPL controller is dependent on the IPL control block for the results of power-on self tests executed for IPL devices. Loading of the IPL record and the code by the IPL ROM does not overwrite the IPL control block. A pointer to the IPL control block is passed to the loaded code. Loaded software can relocate the IPL control block and add entries for IPL devices, but should preserve the rest of the IPL control block. The IPL control block must be intact in order for the ROM warm IPL to work and loaded software must pass ROM a pointer to the IPL control block.

The following shows some of the information that is stored in the IPL control block:

- NVRAM tests results
- Actual IPL device
- Service IPL flag
- Pointer to ROM entry point table
- Pointer to IPL record
- IPL ROM date stamp (IPL ROM build date)
- POST results (a unique structure for each POST)
- Results of expansion code CRC test
- A pointer to a memory bit map
- Pointer to the bad block map
- ROM part number and ID
- An area reserved for future use by IPL ROM.

IPL Record

The IPL record is located in a predefined area on all devices. The record formats are the same for all devices. The IPL ROM loads the IPL record into a known location in RAM. The record is 512 bytes long and contains the following:

- A unique ID to identify the record as an IPL record
- A description of the media: for example, device characteristics
- Descriptions: for example, location, length, and entry point, of one or more code areas to be loaded.
  - The primary load description describes how to load the normal operating system if the operating system is present on the device. If it is not present, the length field of the primary load description must be 0.
  - The alternate load description describes how to a load an alternate operating environment, such as diagnostics, if the alternate operating environment is present on the device. If it is not present, the length field of the alternate load description must be 0.

Interface to the Loaded Code

The IPL ROM loads code into memory and passes the pointer to IPL control block in general purpose register 3.

NVRAM

All machines have NVRAM as described in "NVRAM" on page 4-5.

The following are read from NVRAM by ROM IPL code:

- Check stop count (stored by hardware)
- Device lists stored by software (trusted and service)
- Cyclic redundancy check (CRC) values for the areas of NVRAM from which data is read by the IPL ROM.

LED Operation

ROM displays the appropriate values in the LEDs before executing hardware tests so that if the POST does not return to ROM, the appropriate value is displayed as follows:

1. At the start of each POST, the LEDs are set to the value for that POST.
2. If the POST completes correctly, the next POST is started. Some POSTs execute so quickly that if no error occurs, the display of the corresponding value is not visible to the operator.
3. If the POST code does not complete correctly, the POST LED value remains displayed indicating the error.
4. If the POST detects an error, the sequence controller determines by way of the return code whether the error is a fatal or nonfatal error.
5. If the error is nonfatal, the error information is preserved in the IPL control block, and the sequence controller continues.
6. If the POST error is fatal, the LEDs display an appropriate value steadily, and operation of the system halts.

Errors

Errors occurring during IPL ROM execution can be fatal or nonfatal. The fatal errors are those that prevent an IPL. Nonfatal errors are those that leave the machine in a state to initiate an IPL. The operating system can interrogate the IPL control block to determine if errors occurred during IPL ROM execution.

ROM LED Values During IPL

ROM has been assigned a LED range of 200 to 299. Specific values are assigned during code development. There are special cases where a series of informational data should be presented in the LEDS. Refer to the problem solving section of the product-specific operator's guide for more information on ROM LED values.

The LED codes are displayed during execution of the IPL ROM. Refer to the problem solving section of the product-specific operator's guide for a list of the LED codes.

ROM Entry Point Table

The IPL control block contains a pointer to the ROM entry point table. The ROM entry point table contains the entry point for the ROM warm IPL.

Error Codes

For the list of system error codes, refer to the problem solving section of the product-specific operator's guide.

Index

E

F

Q

R

S

T

U

V

Z