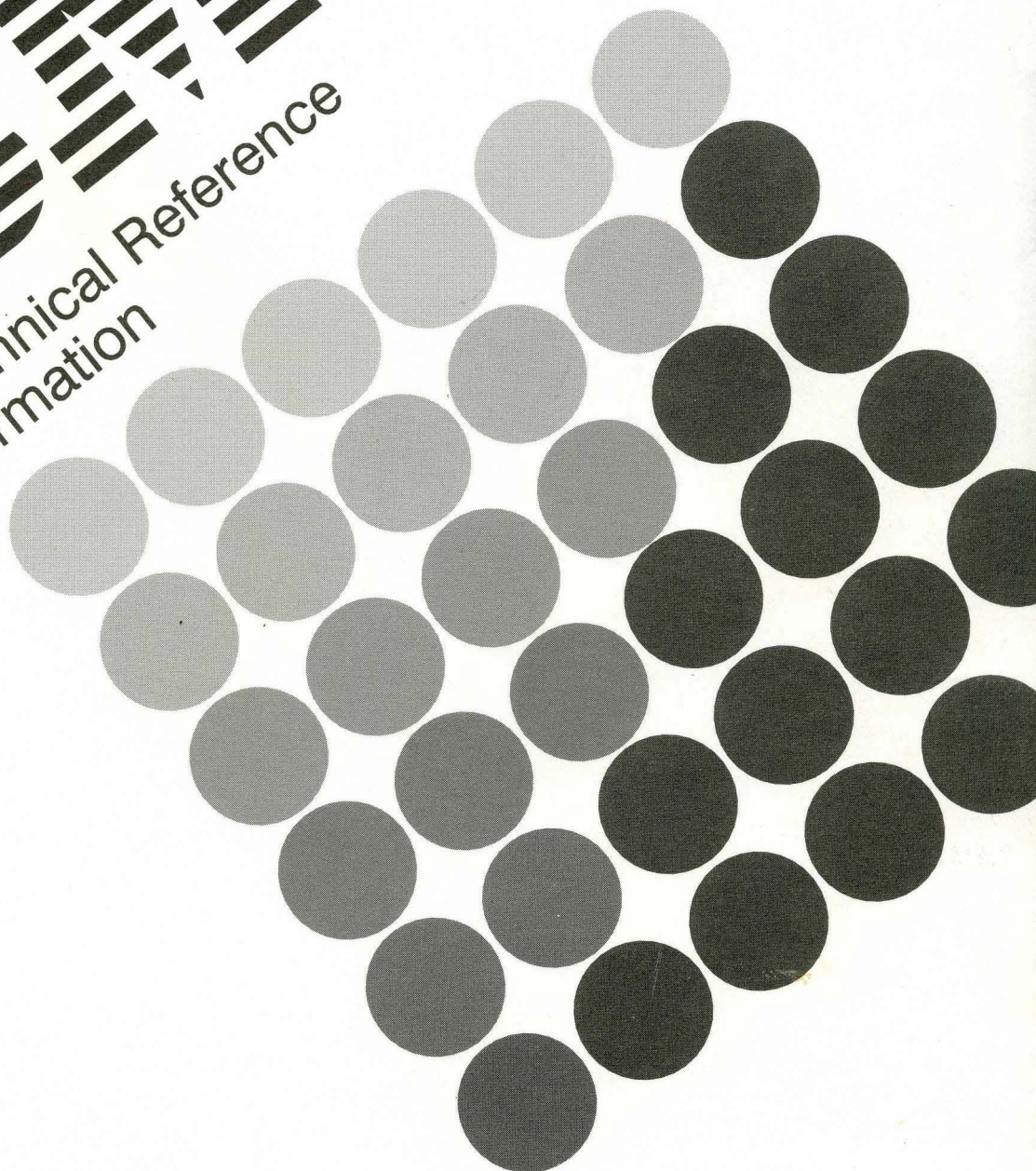


RISC System/6000™  
POWERstation and  
POWERserver

Hardware Technical Reference  
General Information

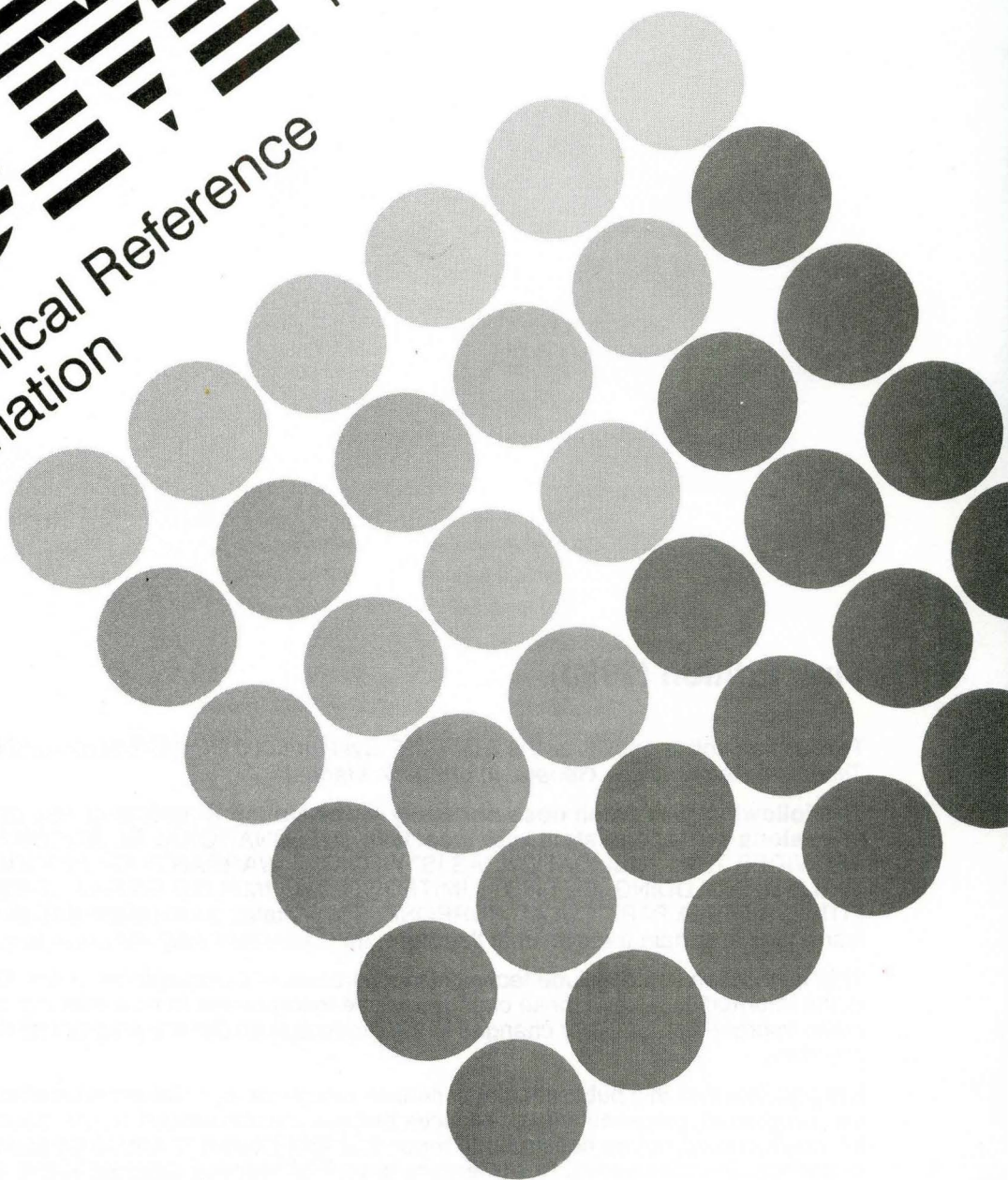






Hardware Technical Reference  
General Information

RISC System/6000™  
POWERstation and  
POWERserver





## First Edition (1990)

This edition notice applies to the *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — General Information Manual*.

**The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country. Any reference to an IBM licensed program in this publication is not intended to state or imply that you can use only IBM's licensed program. You can use any functionally equivalent program instead.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

©Copyright International Business Machines Corporation, 1990. All rights reserved.

Note to US Government Users – Documentation and programs related to restricted rights – Use, duplication, or disclosure is subject to the restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.



---

## Trademarks

The following trademarks apply to this book:

- IBM is a registered trademark of International Business Machines Corporation.
- Personal System/2 and PS/2 are trademarks of International Business Machines Corporation.
- RISC System/6000 is a trademark of International Business Machines Corporation.
- AIX is a trademark of International Business Machines Corporation.







---

# About This Book

## Purpose

The *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — General Information Manual* is one part of the six-part RISC System /6000 hardware technical reference manual. This manual should be used in conjunction with the following RISC System /6000 hardware technical reference manuals:

- *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Options and Devices* (SA23–2646)
- *IBM RISC System/6000 Hardware Technical Reference — 7012 POWERstation and POWERserver* (SA23–2660)
- *IBM RISC System/6000 Hardware Technical Reference — 7013 and 7016 POWERstation and POWERserver* (SA23–2644)
- *IBM RISC System/6000 Hardware Technical Reference — 7015 POWERserver* (SA23–2645)
- *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture* (SA23–2647).

## Audience

The information in this manual is for reference. It is intended for hardware and program designers, programmers, engineers, and anyone else who needs to understand the operation of the IBM RISC System/6000.

## Related Information

- *PS/2 Monochrome Display 8508 Technical Reference* (SA23–2448)
- *60/120MB Fixed-Disk Drive Technical Reference* (S68X–2314)
- *PS/2 5.25-inch External Disk Drive Technical Reference* (S68X–2348)
- *4-Port Multiprotocol Interface Adapter Technical Reference* (S33F–5337)
- *X.25 Co-Processor/2 Technical Reference* (S16F–1879)
- *3270 Emulation Adapter Technical Reference* (GA23–0339).







# Table of Contents

<b>Chapter 1. Introduction to the RISC System/6000 System</b> .....	<b>1-1</b>
Description .....	1-3
Central Electronics Complex .....	1-3
Workstation Hardware .....	1-6
SGR 2564 Processor Chip Set .....	1-9
SGR 2032 Processor Chip Set .....	1-16
 <b>Chapter 2. RISC System/6000 Processors</b> .....	 <b>2-1</b>
Description .....	2-5
Document Conventions .....	2-5
Systems Overview .....	2-6
Instruction Formats .....	2-7
Memory Addressing .....	2-14
Branch Processor .....	2-16
Supervisor Linkage Instruction .....	2-23
Trap Instructions .....	2-24
Condition Register Field Instruction .....	2-25
Condition Register Logical Instructions .....	2-25
Fixed-Point Processor Registers .....	2-29
Fixed-Point Processor Instructions .....	2-31
Fixed-Point Store Instructions .....	2-37
Fixed-Point Load with Update Instructions .....	2-42
Fixed-Point Store with Update Instructions .....	2-46
Fixed-Point Move Assist Instructions .....	2-49
Fixed-Point Address Computation Instructions .....	2-53
Fixed-Point Arithmetic Instructions .....	2-54
Fixed-Point Compare Instructions .....	2-65
Fixed-Point Logical Instructions .....	2-67
Fixed-Point Rotate and Shift Instructions .....	2-73
Floating-Point Processor Overview .....	2-91
Floating-Point Data Representation .....	2-97
Floating-Point Exceptions .....	2-103
Floating-Point Resource Management .....	2-111
Floating-Point Execution Models .....	2-111
Floating-Point Processor Instructions .....	2-114
 <b>Chapter 3. Memory</b> .....	 <b>3-1</b>
Virtual Memory .....	3-3
System Memory .....	3-3



<b>Chapter 4. System I/O Structure</b> .....	<b>4-1</b>
Description .....	4-3
Bit and Byte Numbering Conventions .....	4-9
I/O Bus Protocols .....	4-15
Programming Model .....	4-23
Special Facilities .....	4-70
System I/O and Standard I/O .....	4-78
Exception Reporting and Handling .....	4-80
Implementation Details .....	4-80
 <b>Chapter 5. Vital Product Data</b> .....	 <b>5-1</b>
Description .....	5-3
Keyword Descriptor Summary .....	5-5
Hardware VPD Descriptor Summary .....	5-10
Micro Channel Adapter Requirements .....	5-13
Sample Layout of the Micro Channel Adapter VPD .....	5-17
 <b>Chapter 6. Initial Program Load (IPL) ROM</b> .....	 <b>6-1</b>
Description .....	6-3
IPL ROM Components .....	6-6
IPL ROM Functional Characteristics .....	6-14
Error Codes .....	6-18
 <b>Chapter 7. Keyboard/Tablet/Speaker Adapter</b> .....	 <b>7-1</b>
Description .....	7-5
System Interface: Input/Output Operations to Adapter .....	7-7
Adapter Commands .....	7-14
Adapter Speaker Control .....	7-27
Adapter RAS and Security Functions .....	7-31
Keyboard Device Support Notes .....	7-37
Adapter Design Notes .....	7-37
Adapter and Keyboard Initialization Procedure .....	7-41
Standard I/O Adapter Board to Device Interface .....	7-43
 <b>Chapter 8. Keyboard</b> .....	 <b>8-1</b>
Description .....	8-3
Power-On Routine .....	8-4
Sequential Key-Code Scanning .....	8-4
Commands from the System .....	8-5
Commands to the System .....	8-6
Scan Codes .....	8-7
Clock And Data Signals .....	8-15
Keyboard Character Codes .....	8-17
Shift Status .....	8-22
Speaker .....	8-23
Key Position Layout .....	8-23
Keyboard Layouts .....	8-24
Cables and Connectors .....	8-29
Specifications .....	8-29



<b>Chapter 9. 3-Button Mouse</b> .....	<b>9-1</b>
Description .....	9-3
Operation Modes .....	9-3
Commands .....	9-4
Data Report .....	9-6
Error Handling .....	9-7
Data Frame .....	9-7
Data Transmission .....	9-7
Electrical Interface .....	9-8
Operational Characteristics .....	9-9
Connector Specifications .....	9-9
 <b>Chapter 10. Micro Channel Adapter Support</b> .....	 <b>10-1</b>
Description .....	10-3
IBM Micro Channel Optional Features Supported .....	10-4
Configuration .....	10-5
RISC System/6000 Configuration Procedures .....	10-6
Other Micro Channel Adapter Design Considerations .....	10-6
Adapter Configurations Supported .....	10-7
Dimensions .....	10-7
Power .....	10-9
Micro Channel Architecture Deviations .....	10-10







---

# Chapter 1. Introduction to the RISC System/6000 System

## Chapter Contents

Description .....	1-3
Central Electronics Complex .....	1-3
Workstation Hardware .....	1-6
SGR 2564 Processor Chip Set .....	1-9
Fixed-Point Unit .....	1-9
Floating-Point Unit .....	1-10
Instruction Cache and Branch Processing Unit .....	1-10
Data Cache Unit .....	1-11
Memory Control Unit .....	1-12
I/O Unit .....	1-12
SGR 2564 Processor Pipeline .....	1-14
SGR 2032 Processor Chip Set .....	1-16
RISC System/6000 Table Top Model .....	1-17







---

## Description

The RISC System/6000 unit is a second-generation RISC machine. Like earlier RISC processors, the RISC System/6000 unit 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 RISC processors, however, the RISC System/6000 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 RISC System/6000 unit combines the simplicity of a RISC instruction set with sophisticated hardware design techniques to achieve a short cycle time and a low cycles-per-instruction (CPI) ratio. In a single cycle, four instructions can be run simultaneously: a branch instruction, a fixed-point instruction, a floating-point instruction 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.

---

## Central Electronics Complex

The RISC System/6000 SGR 2564 and SGR 3064 processor chip sets central electronics complex (CEC) contains 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 and output unit (IOU), and a clock chip (CLK). Every memory board contains two data multiplexing modules and one control module for interleaving. The SGR 2564 and SGR 3064 processor chip sets share the same architecture. In this manual, SGR 2564 is used and applies to both the SGR 2564 and SGR 3064 processor chip sets. A block diagram of the SGR 2564 and SGR 3064 processor chip sets is illustrated in Figure 1 on page 1-4.



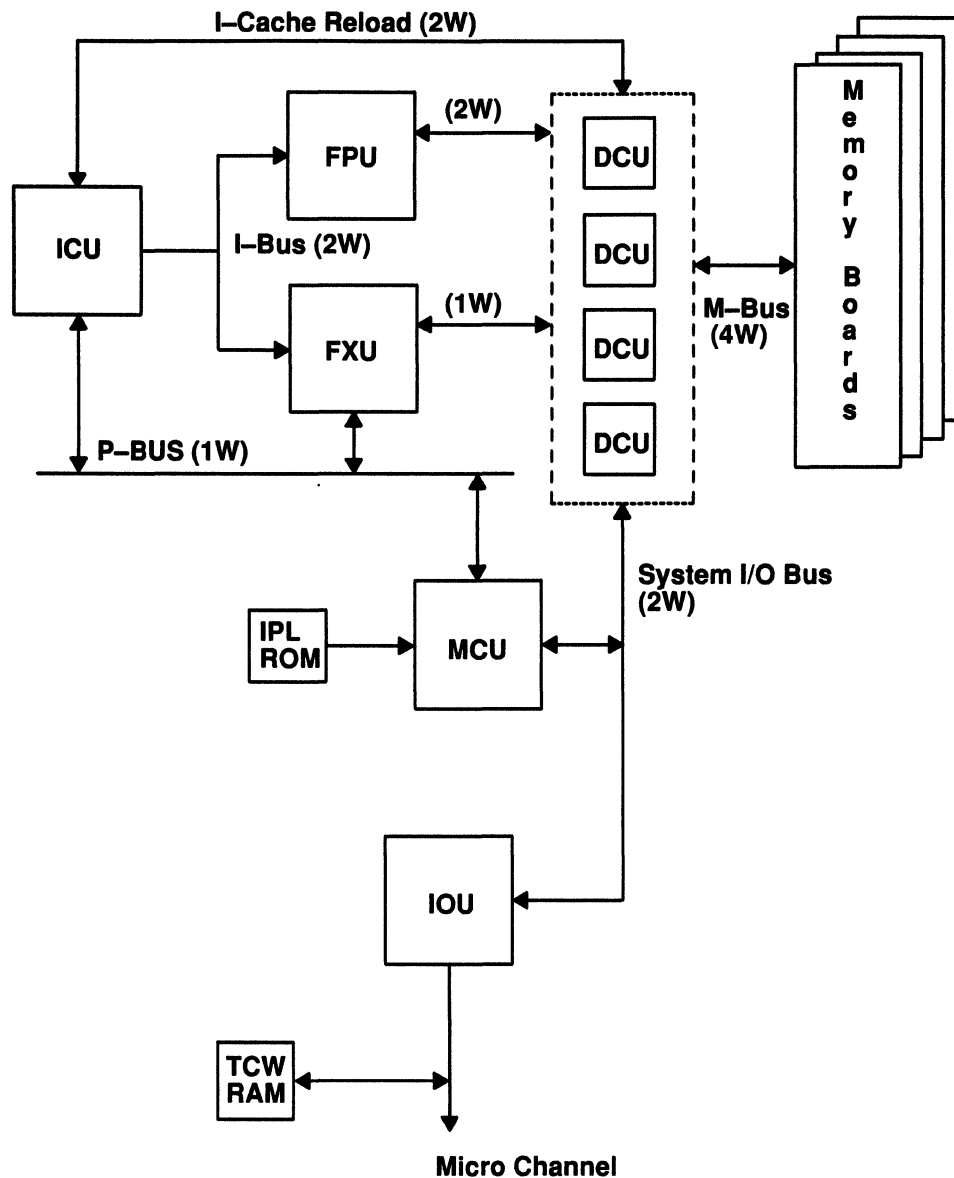


Figure 1. SGR 2564 and SGR 3064 Processor Chip Sets

The ICU contains a two-way set-associative 8K-byte instruction cache. It runs branch instructions, Condition register logical instructions, and supports interrupts. In most 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 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. 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 unit, 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 both fixed-point and floating-point load and store instructions.



Four DCUs provide a four-way set-associative 64K-byte data cache, and form a four-word interface to memory, a two-word interface to FPU, and a single-word 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 I-cache (instruction 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 (Input/Output) 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 1-4 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 (IOCC) that generates the Micro Channel interface. The IOCC uses the data stored in translation control word (TCW) and tag tables for address translation and data protection during I/O operations.



## Workstation Hardware

The RISC System/6000 desktop and rack models have a processor board with a processor chip set and up to eight memory board connectors. The models with the SGR 2564 chip set require that the memory boards to be installed in pairs. On models containing the SGR 2032 chip set, memory boards do not have to be installed in pairs. These models have separate I/O Boards with eight Micro Channel slots and separate Standard I/O Boards as shown in Figure 3 on page 1-8.

The table top RISC System/6000 models have a processor board with a SGR 2032 chip set. The processor board plugs into the connectors on the system board. The system board also has two memory board connectors and four Micro Channel slots as shown in Figure 6 on page 1-17.

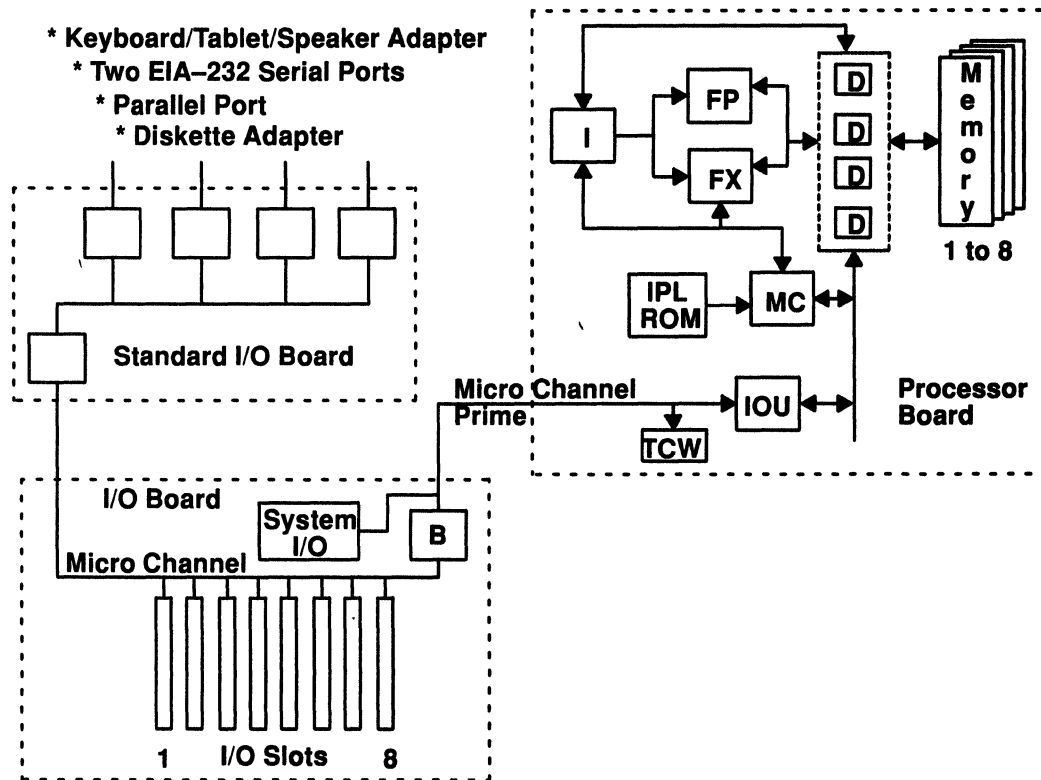


Figure 2. RISC System/6000 Desktop and Rack Organization

The Micro Channel prime interface from the processor board, shown in Figure 2, is attached to the I/O Board where it is buffered (B) and feeds eight Micro Channel I/O slots. These I/O slots can be occupied by Micro Channel boards such as file adapters, tape drive adapters, LAN adapters (Ethernet or Token Ring), display and graphics adapters, coprocessors, terminal emulators, and printer adapters. The I/O Board also contains the system I/O functions. One system I/O function is the On Card Sequencer (OCS) microcontroller, which initializes the processor chip set during IPL and controls the built-in self test (BIST) sequence. Other system I/O functions on the I/O Board are nonvolatile random access memory (NVRAM) for configuration and error logging, operator panel interface for error display, time-of-day clock, computer reset register, and system status and configuration registers. The Standard I/O Board contains the interfaces and connectors to keyboard, mouse, tablet, parallel printer port, diskette, and two EIA-232 serial ports. See the specific system manual for the interfaces and connectors supported.



Figure 3 on page 1-8 shows the physical layouts of the processor board, I/O Board, and Standard I/O Board. Shown on the processor board are the floating-point unit (FP), fixed-point unit (FX), instruction cache unit (I), four data cache units (D), memory control unit (MC), and one or two I/O units (IOU). In addition, the clock chip (CLK), and IPL ROM are also shown. The clock chip has several crystal oscillators around it that vary in speed depending upon the processor chip set. Five 1M-bit dynamic random access memory (DRAM)s that make up the translation control word (TCW) and tag memory are shown at the lower right hand corner. They are used by IOCC for address translation and data protection during I/O operations. Eight memory slots are shown on the right. The IPL ROM is next to the MC chip.

The processor board also carries some Vendor Technology Logic (VTL) parts. The two multiplexers (Mux) shown below the IPL ROM are used to multiplex 16 interrupt lines from the I/O Board to 4 I/O unit inputs. The 64K bytes by 8 OCS ROM and two accompanying latches are at the lower right corner. This ROM holds the test data for the On Card Sequencer (OCS), which resides on the I/O Board, and the latches are used to multiplex and demultiplex the address and data lines.



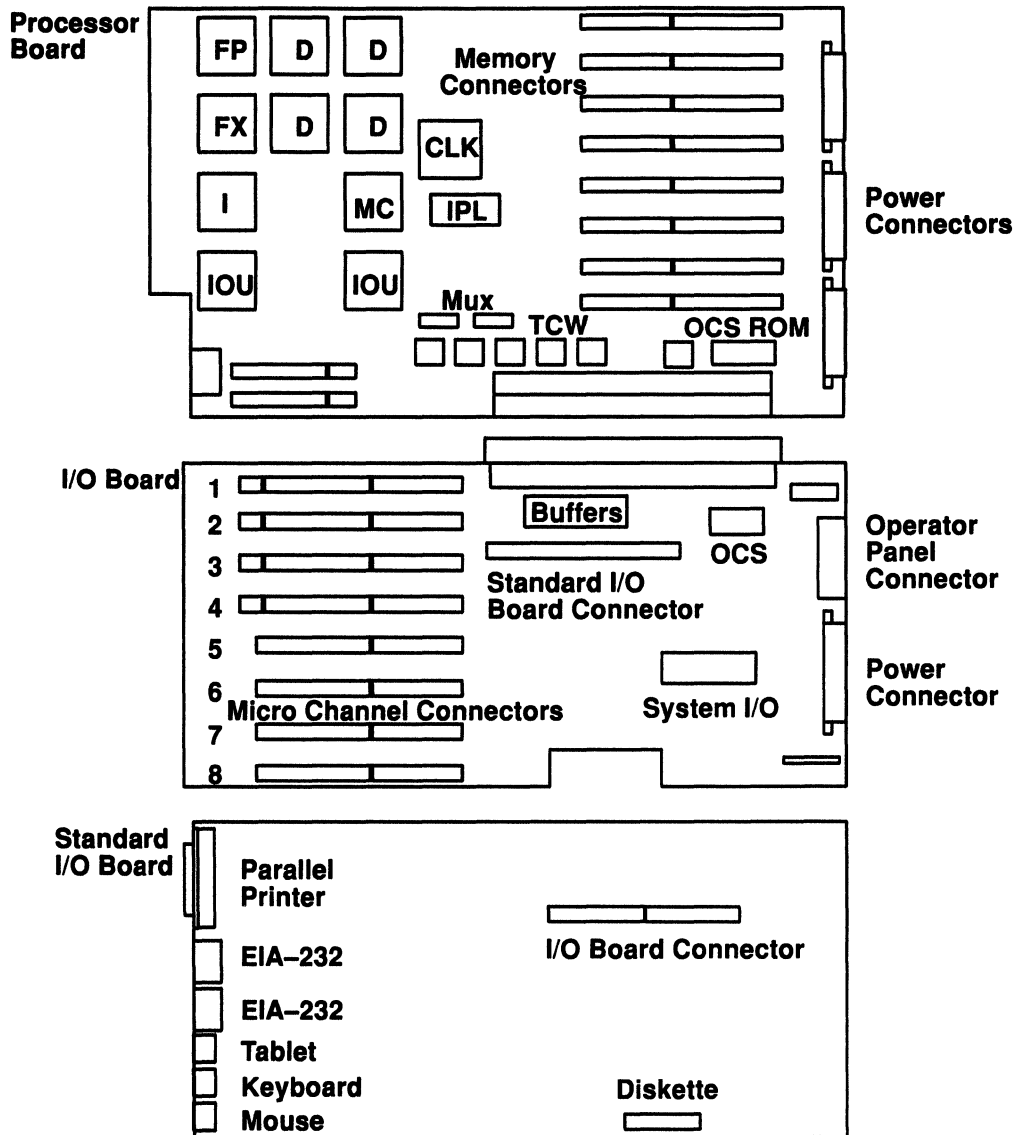


Figure 3. RISC System/6000 Deskside and Rack Processor board, I/O Board, and Standard I/O Board.

The processor board carries a host of tie-up and tie-down resistors, and decoupling capacitors not shown in Figure 3. There are also electromagnetic compatibility (EMC) connectors that couple the chassis ground to board ground in order to minimize the radio-frequency interference (RFI). Power connectors are shown at the right, and the I/O Board connector is at the lower right corner of the processor board.

The I/O Board is placed next to the processor board, and is attached to it by way of a connector as shown in Figure 3. The I/O Board contains eight I/O slots and provides a connector to the operator panel seven-segment light emitting diodes (LEDs). The I/O Board holds the OCS, system I/O, and a collection of additional VTL parts to implement its functions.

The Standard I/O Board fits right behind the I/O Board, and is attached to it through a connector shown in Figure 3. The Standard I/O Board provides interfaces and connectors to keyboard, mouse, tablet, parallel printer port, diskette, and two EIA-232 serial ports.



---

## SGR 2564 Processor Chip Set

As mentioned earlier, the SGR 2564 processor chip set implementation is partitioned into six different semi-custom designed Very Large Scale Integration (VLSI) chips. The features of the chips are summarized in the following subsections.

### Fixed-Point Unit

FXU decodes and runs all fixed-point instructions and floating-point load and store instructions. Both fixed- and floating-point instructions go to the I-buffers of FXU and FPU, and are run concurrently in FXU and FPU. In addition, FXU contains the address translation, data protection, and D-cache directory units.

Its functions include:

- Instruction decode. (Contains four instruction prefetch and two decode buffers.)
- FXU and FPU synchronization logic.
- Real-time clock and decremter facilities.
- Controls for floating-point load and store operation. Address generation and data cache controls for floating-point load and store instructions are generated by FXU.
- Register-to-register (RR) operations. The FXU has a register file that holds thirty-two 32-bit general purpose registers. The register file has five ports. Three ports are read ports and two are write ports (3R,2W). The five ports can all be read and written simultaneously. The hardware associated with the register file implements full bypass (register forwarding) to eliminate hold-offs when two dependent operations (ops) follow each other, and performs register tag allocation so that load operations do not hold off the RR-ops as long as there are no dependencies.
- Instruction runs. RR ops, fixed and floating load and store operations, interrupts, string and character ops, and I/O load and store operations.
- Arithmetic-logic unit, shifter, and rotator.
- Fixed-point multiply and divide operations implemented in hardware. Multiply takes 3 to 5 cycles and divide takes 19 to 20 cycles.
- Address translation unit. Two-way set-associative TLBs with 64 entries in each set.
- Segment registers. Sixteen 32-bit segment registers.
- Hardware TLB reloads. TLB misses are serviced by hardware that has significant performance advantages over other RISC implementations where TLBs are reloaded by software. FXU searches the Hash Anchor table (HAT) and Page Frame table (PFT), and updates the PFT as required.
- Data protection. Page protection and data locking are implemented in hardware.
- Address translation for I-cache TLB reloads. When there is a TLB miss in ICU, FXU brings the PFT entry from the memory, sends it to ICU over the P-bus, and performs the required PFT updates.
- Data cache control, directories, and least recently used (LRU) hardware contain a four-way set associative data-cache directory with 128 entries in each set.
- Store buffers. Data and address of one fixed-point store instruction can be held in this buffer waiting for a convenient time to be put into the D-cache. In addition, there is a four-entry pending store queue for floating-point store instructions.



- Running floating-point load and store instructions.
- Request generation for data cache reload operations.
- Data cache operations such as cache line flush and cache line invalidate.

## Floating-Point Unit

Unlike typical floating-point co-processor chips, the Floating Point Unit (FPU) is tightly coupled with the rest of the processor chip set. FPU and FXU are equal-priority and independent functional units. They receive the instructions from ICU at the same time and run them concurrently. At a given cycle, a fixed- and floating-point instruction can be run simultaneously. FPU has a full 64-bit double-precision data flow, runs floating-point arithmetic ops (multiply, add, divide, subtract), performs conversion between single and double precision, and synchronizes on floating-point load and store operations. FPU conforms to IEEE 754 binary floating-point standard with software support and performs IEEE 64-bit double-precision operations.

The FPU functions include:

- Accumulate instruction ( $A \times B + C$ ) is the key feature of the FPU. The multiply and add operation is run with a single round and with the same delay as a multiply or an add. This reduces the instruction path length by combining two instructions into one and provides exceptional floating-point performance. Due to the 64-bit data flow, the FPU can run a double-precision multiply, add, or accumulate every cycle. The multiply-add operation, by only rounding the final result and producing the full 105 bit intermediate product, provides significantly enhanced precision.
- Register renaming is used to increase the overlap of the running of floating- and fixed-point functional units. This allows floating-point load and store operations to be run independently from the floating-point arithmetic operations and makes it possible to carry on load operations to a target register of a floating-point instruction while the floating-point operation is still going on. This is done by remapping the target register to one of the remap registers. As a result, the FXU can perform floating-point load operations without having to wait for previous floating-point arithmetic operations to be completed.
- Thirty-two architected 64-bit floating-point registers, six rename registers, and two divide registers.
- Hardware divide.
- The leading zero anticipator avoids the full delay of a leading zero detector. This provides overlap of addition and normalization.

## Instruction Cache and Branch Processing Unit

The ICU contains a two-way set associative 8K-byte I-cache with a line size of 64 bytes. The ICU processes branch instructions and Condition register (CR) logical instructions. Then, it removes them from the instruction stream and dispatches the rest of the instructions to fixed- and floating-point units. In most cases, fixed- and floating-point units receive an uninterrupted instruction sequence and do not see the effect of the branches. This is referred to as zero-cycle branches. Usually, unconditional branches cause no delay in the pipeline. Conditional branches that are not taken (fall-through) also have no penalty because ICU dispatches the branch-not-taken path to FXU and FPU before figuring out the outcome of the branch. Of course, the branch-not-taken path instructions are cancelled if the conditional branch is taken. The branch-taken path is fetched from the I-cache arrays but is not dispatched to FXU and FPU. Conditional branches that are taken may delay the pipeline by 0 to 3 cycles depending on how much earlier the Condition register was set.



The compiler tries to move the condition code setting instruction far enough ahead of the conditional branch to minimize the conditional-branch penalty.

The ICU performs the following functions:

- Instruction caching. Contains a two-way set associative 8K-byte cache, directories, and hardware to support a Least-Recently-Used (LRU) replacement algorithm.
- Instruction address translation. Contains a two-way set associative translation look-aside buffer (TLB) with 16 entries in each set.
- Instruction fetching. A maximum of four instructions can be fetched from the cache arrays in a single cycle.
- Instruction dispatching. Dispatches a maximum of four instructions per cycle: two instructions internally to branch and condition-register units and two instructions externally to FXU and FPU.
- Branch run with zero-cycle branches.
- Condition register logical instruction run.
- Interrupt control.
- Manipulation of architected registers.

## Data Cache Unit

The SGR 2564 chip set has a four-way set associative 64K-byte of data cache divided into four data cache chips of 16K-byte each. The cache-line size is 128 bytes and the cache is implemented as a store-back cache to minimize the memory bus traffic. (When the data is stored in the D-cache, it is not sent to memory. The data is written into memory only when a dirty line is replaced.) DCU supports fixed- and floating-point load and store operations, and provides a path from memory for I-cache reload and DMA operations. D-cache provides bit steering and ECC for load and store, I-cache reload, DMA, and memory scrub operations. D-cache directories, LRU hardware, dirty-bit information, and TLBs are in the FXU.

The main features of the DCU include:

- The collection of four D-cache chips has a four-word interface to system memory for high-bandwidth cache reload and store-back operations.
- Separate data interfaces to FXU (1 word) and FPU (2 words).
- D-cache reload buffer (CRB). A 128-byte CRB implemented across the four DCUs receives data from memory, IPL, FXU, and FPU. A load operation can read data from CRB if the data is from a line that is not yet loaded to cache arrays but is in the CRB. A fast load-through path that bypasses the cache arrays is provided from the memory bus to the FXU and FPU to minimize the load operation delays. Unlike simpler cache implementations, which do not have a CRB, the SGR 2564 processor chip set does not have to wait for the entire cache line to be brought from memory before it can access the data required by the load instruction that caused the cache miss. This makes long cache lines practical, which in return improves the D-cache hit ratio.
- Store-back buffer (SBB). A 128-byte SBB implemented across the four DCUs accepts data from D-cache array or directly from CRB and passes it to system memory. Store-back buffers improve the performance because the data cache arrays are not kept busy during the store-back sequence. The entire line is loaded in parallel into the SBB, and the data is sent to the memory over the memory bus in 8 cycles. The DCU can service the processor chip set during these cycles because the arrays are freed up by



SBB. In addition, the store-back data can be left in the SBB and stored back later if a higher priority memory access is pending.

- I-cache reload buffer (IRB). This receives data from memory or IPL ROM, and sends it to the I-cache. The data from system memory is processed through ECC and bit-steering logic. This buffer is also used for memory scrubbing.
- I/O DMA buffer (IOB). Buffers the data between system memory and I/O. The DMA traffic goes between DCU and IOCC by way of the system I/O bus.
- ECC (single-bit correct, double-bit detect) and bit-steering logic for incoming and outgoing data from and to memory including D- and I-cache reload, DMA, and memory scrub operations.

## Memory Control Unit

The memory control unit (MCU) is the central system controller. The MCU controls the interface between D-cache and system memory, oversees DMA operations between memory and the IOCC, provides a data path for I/O loads and stores between the processor chip set and IOCC, forms an interface to the IPL ROM, and controls memory scrub operations.

The main features of the MCU are:

- Drives all control lines to memory.
- Controls DMA operations between IOCC and system memory.
- Controls memory interface to DCU. MCU informs DCU where the incoming data should go. The MCU also directs the unloading of DMA and I-cache buffers.
- Controls the memory scrubbing. MCU generates the addresses and records any memory errors DCU detects.
- Controls reading and writing of bit-steering registers.
- Contains the Bank Configuration registers, which indicate the size and starting point of each bank of system memory.
- Provides a data-path for I/O load and store operations between the processor chip set and IOCC.
- Performs arbitration for the memory bus.
- Provides an interface to initial program load read-only memory (IPL ROM).
- Collects external interrupts from the IOCC, decremter, power supply, and system memory.

## I/O Unit

The I/O unit (IOU) contains an I/O channel controller (IOCC) that generates the Micro Channel Prime interface. The data interface between the processor/system memory and the I/O unit is by way of the two-word wide system I/O bus. The Micro Channel has a one-word address bus and a one-word data bus. The IOCC supports an I/O architecture geared for performance, robustness, and error recoverability. The Micro Channel architecture supports streaming data, address and data parity, and synchronous exception reporting functions (I/O load and store commands cause precise interrupts like regular load and store commands). The main function of the IOCC is to transfer data between system memory and adapters on the Micro Channel. The processor unit can transfer data to and from the adapters using I/O load and store operations, and the adapters can transfer data to and from system memory using DMA. The IOCC supports both DMA bus masters and DMA slaves. All data transfers



support address protection mechanisms to provide data security. Up to 15 DMA channels and 16 levels of interrupts are supported by the IOCC. With the new streaming data mode, multiple data cycles can be transferred within one bus envelope. This amortizes device selection overhead across the entire packet and nearly doubles the performance for large data bursts. Precise I/O load and store interrupts improve error recoverability.

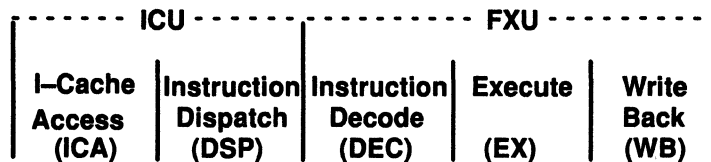
The main features of the IOCC include:

- Interface to System I/O bus and Micro Channel.
- Programmed I/O (PIO) operations to and from the following address spaces.
  - System memory space
  - Micro Channel I/O space (I/O adapters)
  - Micro Channel memory space (memory on the Micro Channel)
  - IOCC space
  - Architected IOCC registers
  - Tag and TCW RAM.
- I/O load and store operations are performed with or without alignment and with a protection mechanism. Protection is provided by TCW for system memory and limit registers for I/O devices.
- Handles data to and from DMA slaves.
- Handles data to and from DMA bus masters.
- Address translation for load and store operations and DMA bus masters.
- Handles I/O interrupts.
- Supports various IOCC commands such as enable and disable DMA, DMA device buffer flush, lock, and time delay.



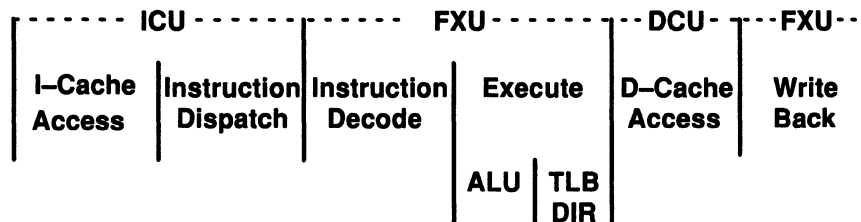
## SGR 2564 Processor Pipeline

Because of the complexity of the pipeline, various instruction buffers, hold-off conditions, and the special cases, there are many possible variations and exceptions in the way an instruction can be run in the RISC System/6000 unit. With that in mind, a typical pipeline for a register-to-register (RR) operation could be constructed as follows:



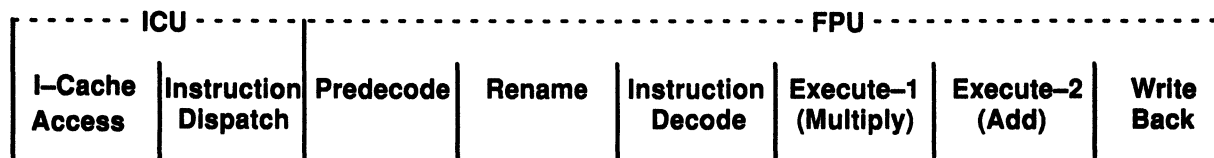
In the first cycle, ICU reads the cache array, then in the dispatch (second) cycle the instruction is partially decoded to see if it is a branch, and non-I-cache instructions are dispatched to FXU and FPU. At the third cycle, FXU decodes the instruction, accesses the register file, and latches up the values read from the register file at the Arithmetic Logic Unit (ALU) input registers. In the execution (fourth) cycle, the ALU operation takes place. Finally, the result is written back into the register file in the fifth cycle.

A typical pipeline for a load is as follows:



In the first half of the execution cycle, the ALU operation takes place and the virtual address is calculated. In the second half of the execution cycle, TLBs are accessed to determine the real page number and, in parallel, the D-cache directories are accessed to see if the data is in the cache. In the fifth cycle, data cache is accessed and the data is shipped back to FXU or FPU where it is latched in a register. And in the sixth cycle, the data is written into the register file.

The floating-point arithmetic operation pipeline is as follows:



There is a synchronization cycle before the decode operation, and the floating-point arithmetic operations (multiply, add, accumulate) take two cycles to run.



Because the RISC System/6000 unit is pipelined, all these operations are overlapped as shown in the following illustration, and all the hardware resources are utilized to their full potential.

Instruction	Cycle							
	1	2	3	4	5	6	7	8
1	ICA	DSP	DEC	EX	WB			
2		ICA	DSP	DEC	EX	WB		
3			ICA	DSP	DEC	EX	WB	
4				ICA	DSP	DEC	EX	WB

As mentioned earlier, the pipeline is not as simple as described in the preceding text because ICU contains I-buffers and can read up to four instructions per cycle from the cache array. I-cache can dispatch two instructions per cycle to FXU and FPU. In addition, both FXU and FPU contain their own I-buffers. ICU looks ahead and runs branches such that they are in effect taken out of the instruction stream.



## SGR 2032 Processor Chip Set

The SGR 2032 processor chip set is a cost-reduced version of the SGR 2564 processor chip set. The SGR 2032 processor chip set is shown in Figure 4.

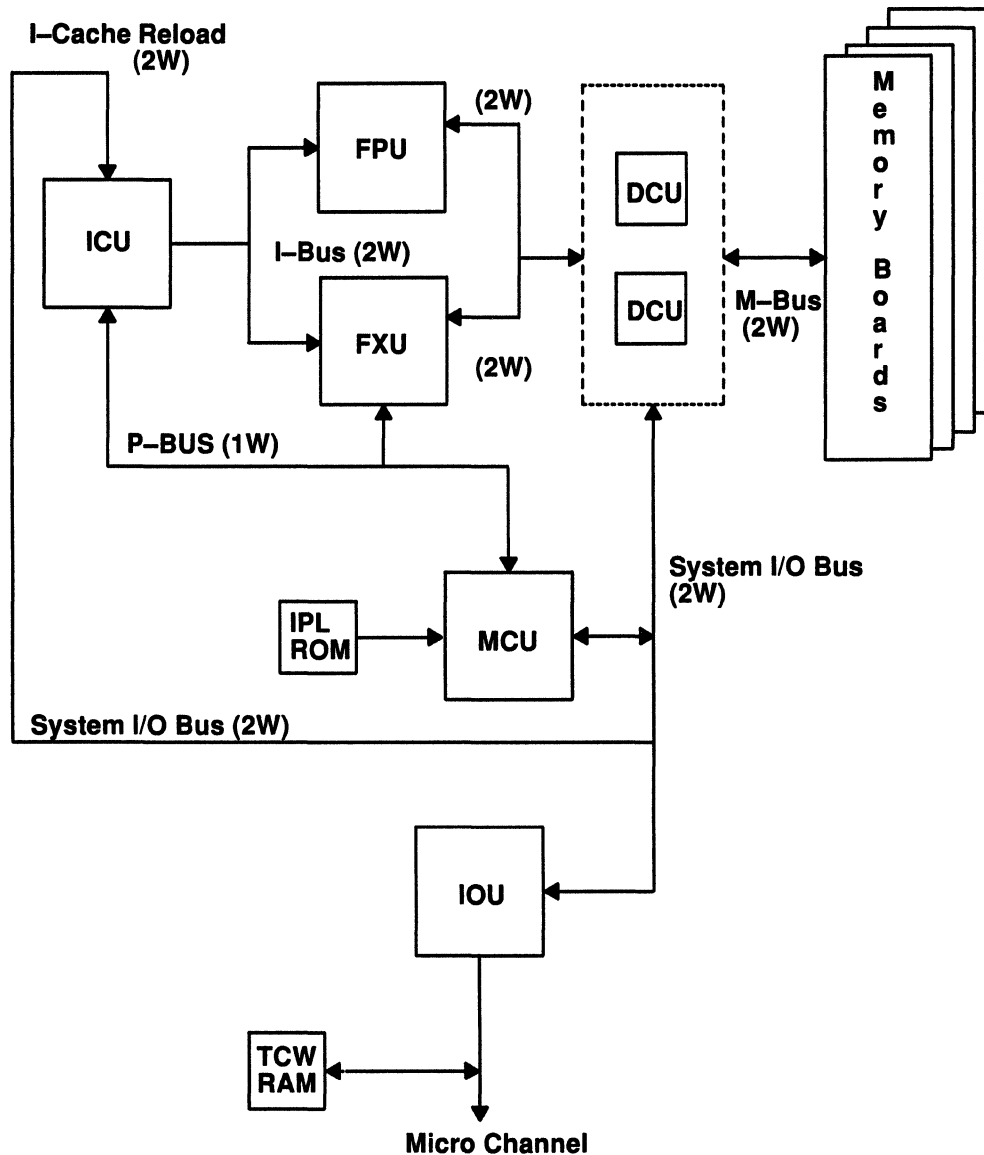


Figure 4. SGR 2032 Processor Chip Set

The major differences between the SGR 2032 processor chip set and the SGR 2564 processor chip set are as follows:

- The SGR 2032 processor chip set has only two DCUs rather than four.
- Fixed- and floating-point data buses are dotted together. DCU provides a two-word bus. Because FXU has only a single-word data interface, it is tied to only half of the bus. DCU manipulates the data accordingly when FXU is using the bus.
- In the SGR 2032 processor chip set, the D-cache line size is 64 bytes (half of the SGR 2564 processor chip set D-cache line size).



- DCU sends the data to reload the I-cache over the system I/O bus rather than having a dedicated I-cache reload bus to ICU.
- The processor chip set has a two-word memory interface rather than a four-word interface. As a result, the SGR 2032 processor chip set requires a minimum of one memory board and the SGR 2564 processor chip set requires a minimum of two memory boards. The minimum memory configuration for the SGR 2032 processor chip set is a single 8M-byte memory board.

The SGR 2032 processor chip set and the SGR 2564 processor chip set use the same chips. There are no new part numbers. A mode pin tells FXU, DCU, and MCU if the system is a SGR 2032 processor chip set or the SGR 2564 processor chip set.

## RISC System/6000 Table Top Model

The RISC System/6000 table top model uses the SGR 2032 processor chip set as shown in Figure 4 on page 1-16. Figure 5 shows the processor board and Figure 6 shows the system board for the RISC System/6000 table top model.

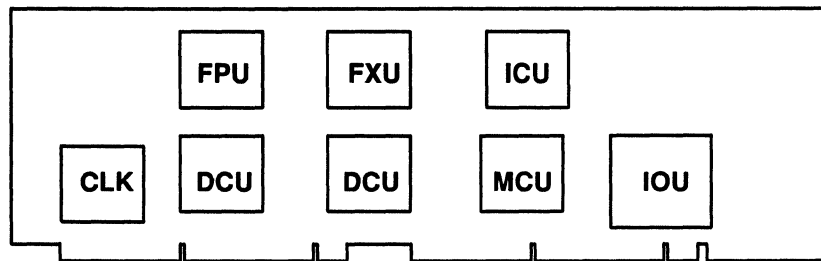


Figure 5. RISC System/6000 Table Top Processor Board

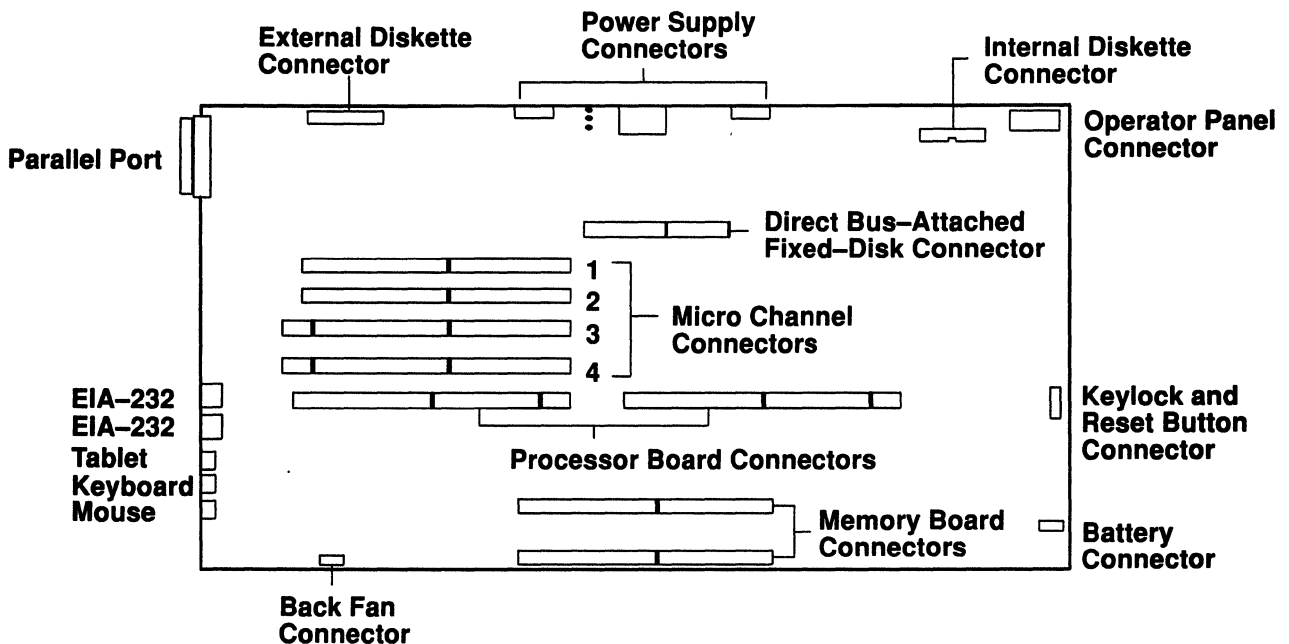


Figure 6. RISC System/6000 Table Top System Board







---

## Chapter 2. RISC System/6000 Processors

### Chapter Contents

Description .....	2-5
Document Conventions .....	2-5
Systems Overview .....	2-6
Instruction Formats .....	2-7
Memory Addressing .....	2-14
Effective Address Calculation .....	2-14
Branch Processor .....	2-16
Branch Processor Registers .....	2-16
Branch Instructions .....	2-20
Supervisor Linkage Instruction .....	2-23
Trap Instructions .....	2-24
Condition Register Field Instruction .....	2-25
Condition Register Logical Instructions .....	2-25
Fixed-Point Processor Registers .....	2-29
General Purpose Registers .....	2-29
Fixed-Point Exception Register .....	2-29
Multiply Quotient Register .....	2-30
Fixed-Point Processor Instructions .....	2-31
Fixed-Point Store Instructions .....	2-37
Fixed-Point Load with Update Instructions .....	2-42
Fixed-Point Store with Update Instructions .....	2-46
Fixed-Point Move Assist Instructions .....	2-49
Fixed-Point Address Computation Instructions .....	2-53
Fixed-Point Arithmetic Instructions .....	2-54
Fixed-Point Compare Instructions .....	2-65
Fixed-Point Logical Instructions .....	2-67
Fixed-Point Rotate and Shift Instructions .....	2-73
Fixed-Point Rotate with Mask Instructions .....	2-73
Rotate Left Immediate Then Mask Insert (M-Form) .....	2-73
Rotate Left Then Mask Insert (M-Form) .....	2-74
Rotate Left Immediate Then AND With Mask (M-Form) .....	2-74
Rotate Left Then AND With Mask (M-Form) .....	2-74
Fixed-Point Rotate Bit Instructions .....	2-75
Rotate Right And Insert Bit (X-Form) .....	2-75
Fixed-Point Bit Mask Instructions .....	2-75
Mask Generate (X-Form) .....	2-75
Mask Insert From Register (X-Form) .....	2-76
Fixed-Point Shift Instructions .....	2-76
Shift Left (X-Form) .....	2-76
Shift Right (X-Form) .....	2-77
Shift Left With MQ (X-Form) .....	2-77
Shift Right With MQ (X-Form) .....	2-78
Shift Left Immediate With MQ (X-Form) .....	2-78
Shift Right Immediate With MQ (X-Form) .....	2-79



Shift Left Long Immediate With MQ (X-Form) .....	2-79
Shift Right Long Immediate With MQ (X-Form) .....	2-80
Shift Left Long With MQ (X-Form) .....	2-80
Shift Right Long With MQ (X-Form) .....	2-81
Shift Left Extended (X-Form) .....	2-81
Shift Right Extended (X-Form) .....	2-82
Shift Left Extended With MQ (X-Form) .....	2-82
Shift Right Extended With MQ (X-Form) .....	2-83
Shift Right Algebraic Immediate (X-Form) .....	2-83
Shift Right Algebraic (X-Form) .....	2-84
Shift Right Algebraic Immediate With MQ (X-Form) .....	2-84
Shift Right Algebraic With MQ (X-Form) .....	2-85
Shift Right Extended Algebraic (X-Form) .....	2-85
Double-Precision Shifts .....	2-86
Move To and Move From System Registers Instructions .....	2-87
Move To and Move From Condition Register Instruction .....	2-89
Move From Machine State Register Instruction .....	2-90
Floating-Point Processor Overview .....	2-91
Floating-Point Registers .....	2-92
Floating-Point Status and Control Register .....	2-93
Floating-Point Data Representation .....	2-97
Data Format .....	2-97
Value Representation .....	2-98
Binary Floating-Point Numbers .....	2-98
Normalized Numbers (+NOR) .....	2-98
Zero values (+0) .....	2-99
Denormalized Numbers (+DEN) .....	2-99
Infinities (+INF) .....	2-99
Not a Numbers (NaNs) .....	2-99
Normalization and Denormalization .....	2-100
Precision .....	2-101
Rounding .....	2-101
Data Handling .....	2-102
Floating-Point Exceptions .....	2-103
Invalid Operation Exception .....	2-105
Definition .....	2-105
Action .....	2-105
Zero Divide Exception .....	2-106
Definition .....	2-106
Action .....	2-106
Overflow Exception .....	2-107
Definition .....	2-107
Resultant Value .....	2-107
Insuring Correct Results .....	2-107
Action .....	2-108
Underflow Exception .....	2-109
Definition .....	2-109
Action .....	2-109
Inexact Exception .....	2-110
Definition .....	2-110
Action .....	2-110



Floating-Point Resource Management .....	2-111
Floating-Point Execution Models .....	2-111
Execution Model for IEEE Operations .....	2-111
Execution Model for Multiply-Add Type Instructions .....	2-113
Floating-Point Processor Instructions .....	2-114
Floating-Point Load Instructions .....	2-114
Normalized Operand .....	2-114
Infinity / QNaN / SNaN / Zero .....	2-114
Denormalized Operand .....	2-114
Load Floating-Point Single (D-Form) .....	2-115
Load Floating-Point Single Indexed (X-Form) .....	2-115
Load Floating-Point Double (D-Form) .....	2-116
Load Floating-Point Double Indexed (X-Form) .....	2-116
Load Floating-Point Single With Update (D-Form) .....	2-117
Load Floating-Point Single With Update Indexed (X-Form) .....	2-117
Load Floating-Point Double With Update (D-Form) .....	2-118
Load Floating-Point Double With Update Indexed (X-Form) .....	2-118
Floating-Point Store Instructions .....	2-119
No Denormalization Required .....	2-119
Denormalized Operand .....	2-119
Store Floating-Point Single (D-Form) .....	2-120
Store Floating-Point Single Indexed (X-Form) .....	2-120
Store Floating-Point Double (D-Form) .....	2-121
Store Floating-Point Double Indexed (X-Form) .....	2-121
Store Floating-Point Single With Update (D-Form) .....	2-122
Store Floating-Point Single With Update Indexed (X-Form) .....	2-122
Store Floating-Point Double With Update (D-Form) .....	2-123
Store Floating-Point Double With Update Indexed (X-Form) .....	2-123
Floating-Point Move Instructions .....	2-124
Floating Move Register (X-Form) .....	2-124
Floating Negate (X-Form) .....	2-124
Floating Absolute Value (X-Form) .....	2-124
Floating Negative Absolute Value (X-Form) .....	2-125
Floating-Point Arithmetic Instructions .....	2-126
Floating Add (A-Form) .....	2-126
Floating Subtract (A-Form) .....	2-127
Floating Multiply (A-Form) .....	2-127
Floating Divide (A-Form) .....	2-128
Floating Round To Single Precision (X-Form) .....	2-128
Floating-Point Accumulate Instructions .....	2-129
Floating Multiply Add (A-Form) .....	2-129
Floating Multiply Subtract (A-Form) .....	2-130
Floating Negative Multiply Add (A-Form) .....	2-131
Floating Negative Multiply Subtract (A-Form) .....	2-132
Floating-Point Compare Instructions .....	2-133
Floating Compare Unordered (X-Form) .....	2-133
Floating Compare Ordered (X-Form) .....	2-134
Floating-Point Status and Control Register Instructions .....	2-135
Move From FPSCR (X-Form) .....	2-135
Move To Condition Register From FPSCR (X-Form) .....	2-135
Move To FPSCR Fields (XFL-Form) .....	2-136



Move To FPSCR Field Immediate (X-Form) .....	2-137
Move To FPSCR Bit 1 (X-Form) .....	2-137
Move To FPSCR Bit 0 (X-Form) .....	2-138
Floating Point Round to Single Model .....	2-139
Floating Round to Single Model: .....	2-139
Disabled Exponent Underflow: .....	2-139
Enabled Exponent Underflow: .....	2-140
Disabled Exponent Overflow: .....	2-141
Enabled Exponent Overflow: .....	2-142
Infinity Operand: .....	2-142
QNaN Operand: .....	2-142
SNaN Operand: .....	2-142
Normal Operand: .....	2-143
Round Single(sign,exp,frac,G,R,X): .....	2-143
RISC System/6000 Instruction Set .....	2-144



---

## Description

This chapter describes the document conventions, a general systems overview, instruction formats, and memory addressing.

---

## Document Conventions

The following conventions are used throughout the RISC System/6000 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
- $n \times b'0'$  means  $n$  zeros
- $n \times b'1'$  means  $n$  ones
- $(RA|0)$  means the contents of register RA if the RA field has the value 1–31, or the value 0 if the RA field is 0
- $(Rx)$  means the contents of register Rx
- $(FRx)$  means the contents of register FRx
- $X(p)$  means bit p of register or field X
- $X_p$  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
- $X_{p-q}$  means bits p through q of register or field X
- $\neg(RA)$  means the one's complement of the contents of register RA
- $/, //, ///, \dots$  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  $n^{\text{th}}$  power
- Field i refers to bits  $4 \times i$  to  $(4 \times i) + 3$  of a register
- Positive means greater than 0
- Negative means less than 0
- Instructions are assumed to be non-privileged 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, described on page 2-20
- Fixed-point processor instructions, described on page 2-31
- Floating-point processor instructions, described on page 2-114.

See Figure 7 for a representation of the logical partitioning provided by the IBM RISC System/6000 architecture. The processing unit is a word-oriented fixed-point processor and in a doubleword-oriented floating-point processor. The RISC System/6000 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).

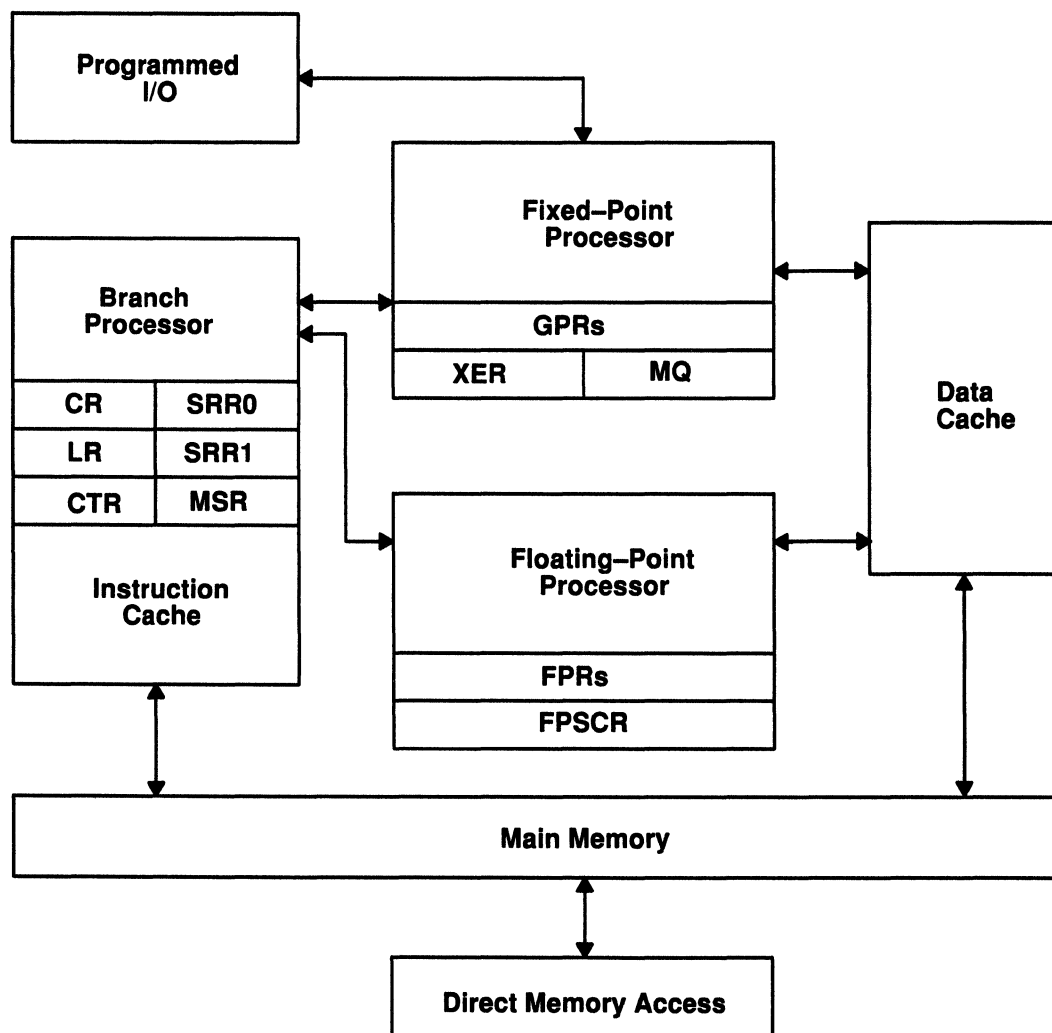


Figure 7. System Architecture View



## 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.

### D Form

0	6	11	16
OPCD	RT	RA	D
	RS		SI
	FRT		UI
	TO		
	BF		
	FRS		

### B Form

0	6	11	16	30	31
OPCD	BO	BI	BD	AA	LK

### I Form

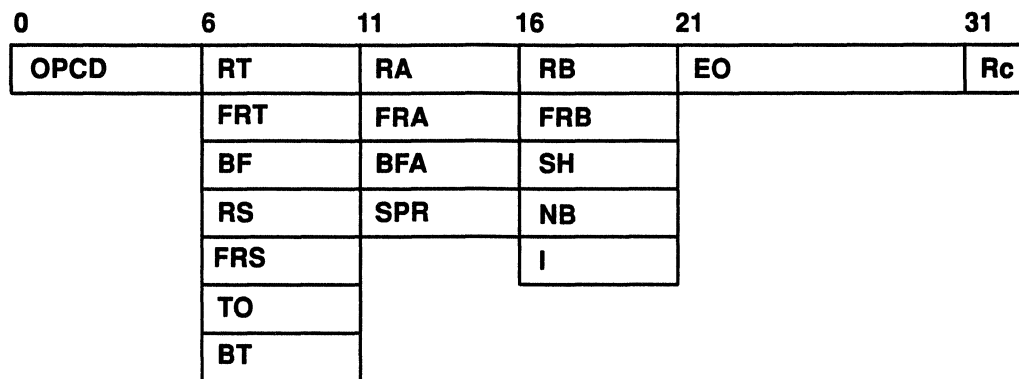
0	6	30	31
OPCD	LI	AA	LK

### SC Form

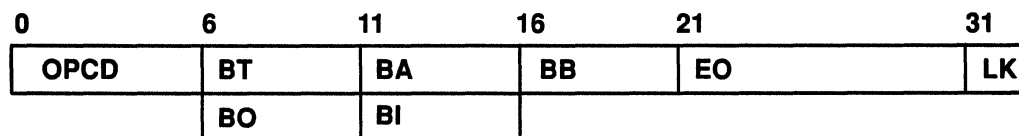
0	6	11	16	20	27	30	31
OPCD	///	///	FL1	LEV	FL2	SA	LK
			SV				



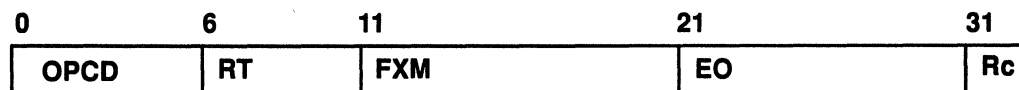
## X Form



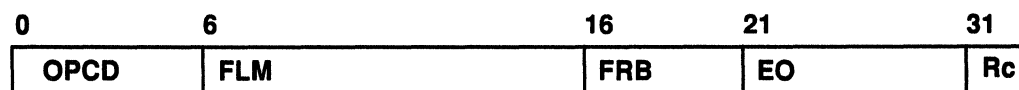
## XL Form



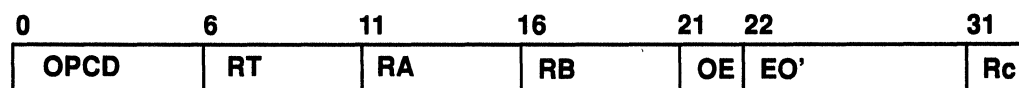
## XFX Form



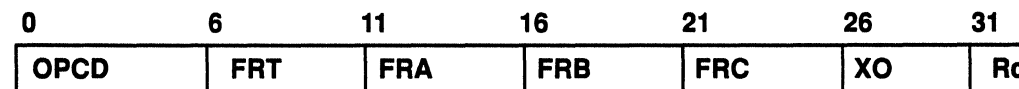
## XFL Form



## XO Form

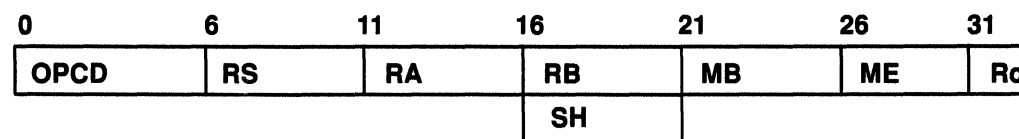


## A Form



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

## M Form



## Instruction Fields

AA (30)

Absolute Address bit



<b>Bit</b>	<b>Description</b>
<b>0</b>	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.
<b>1</b>	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.
<b>BA (11–15)</b>	Field used to specify a bit in the Condition register (CR) to be used as a source.
<b>BB (16–20)</b>	Field used to specify a bit in the CR to be used as a source.
<b>BD (16–29)</b>	Immediate field specifying a 14-bit signed two's complement branch displacement, which is concatenated on the right with b'00' and sign extended to 32 bits.
<b>BF (6–8)</b>	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 $i \times 4$ to $(i \times 4) + 3$ of the register.
<b>BFA (11–13)</b>	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 $j \times 4$ to $(j \times 4) + 3$ of the register.
<b>BI (11–15)</b>	Field used to specify the bit in the CR to be used as the condition of the branch.



**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:

<b>BO</b>	<b>Description</b>
<b>0000x</b>	Decrement the CTR, then branch if the decremented CTR $\neq 0$ and condition is false.
<b>0001x</b>	Decrement the CTR, then branch if the decremented CTR = 0 and condition is false.
<b>001xx</b>	Branch if condition is false.
<b>0100x</b>	Decrement the CTR, then branch if the decremented CTR $\neq 0$ and condition is true.
<b>0101x</b>	Decrement the CTR, then branch if the decremented CTR = 0 and condition is true.
<b>011xx</b>	Branch if condition is true.
<b>1x00x</b>	Decrement the CTR, then branch if the decremented CTR $\neq 0$ .
<b>1x01x</b>	Decrement the CTR, then branch if the decremented CTR = 0.
<b>1x1xx</b>	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 two's complement integer sign extended to 32 bits.

**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).

**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)** 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.



**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:

**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 an load or store string immediate.

**OPCD (0–5)** The basic opcode field of the instruction.

**OE (21)** Used for extended arithmetic to inhibit 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)** Record bit.

Setting	Description
---------	-------------

0	Do not set the Condition register.
---	------------------------------------

1	Set the Condition register to reflect the result of the operation.
---	--

For fixed-point instructions, CR bits (0–3) 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 (4–7) 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.

**SA (30)** 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.



<b>SPR (11–15)</b>	Special Purpose register.	
	<b>SPR</b>	<b>Special Purpose Register</b>
	<b>00000 (00)</b>	<b>MQ</b>
	<b>00001 (01)</b>	<b>XER</b>
	<b>00100 (04)</b>	<b>from RTCU</b>
	<b>00101 (05)</b>	<b>from RTCL</b>
	<b>00110 (06)</b>	<b>from DEC</b>
	<b>01000 (08)</b>	<b>LR</b>
	<b>01001 (09)</b>	<b>CTR</b>
	<b>10100 (20)</b>	<b>to RTCU</b>
	<b>10101 (21)</b>	<b>to RTCL</b>
	<b>10110 (22)</b>	<b>to DEC</b>
	<b>11010 (26)</b>	<b>SRR 0</b>
	<b>11011 (27)</b>	<b>SRR 1.</b>
<b>TO (6–10)</b>	TO bit ANDed with condition.	
	<b>TO bit</b>	<b>ANDed with Condition</b>
	<b>6</b>	<b>Compares less than</b>
	<b>7</b>	<b>Compares greater than</b>
	<b>8</b>	<b>Compares equal</b>
	<b>9</b>	<b>Compares logically less than</b>
	<b>10</b>	<b>Compares logically greater than.</b>
<b>UI (16–31)</b>	Immediate field used to specify a 16–bit unsigned integer.	
<b>XO (26–30)</b>	A 5–bit extended opcode used by A–form instructions.	



---

## 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 8 for an example of the 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 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 2-18.

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.

Doubleword	000								
Word	000				100				
Halfword	000		010		100		110		
Byte	000	001	010	011	100	101	110	111	
	0	8	16	24	32	40	48	56	63

Figure 8. Memory Organization

## 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 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 0–29 of the Link register or the Count register are concatenated on the right with b'00' to form the effective address.



---

## Branch Processor

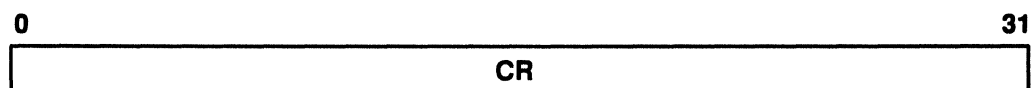
This section describes the registers and instructions that make up the branch processor facilities.

### 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 record bit (Rc) equals a value of 1 in most fixed-point instructions, the CR Field 0 (condition register bits 0–3) is set by a compare of the result to a value of 0. 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 negative or less 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.  |

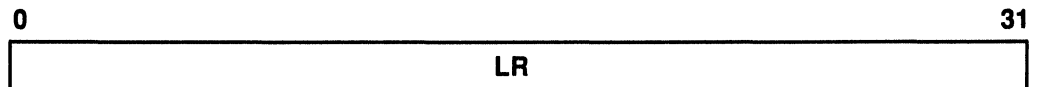
When the Rc bit equals a value of 1 in all floating-point instructions except the floating-point compares, CR Field 1 (condition register bits 4–7) is set to the floating-point exceptions status. These bits are interpreted as shown in the following list:

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(FX) 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 4–7 are copies of bits 0–3 in the Floating-Point Status and Control register.

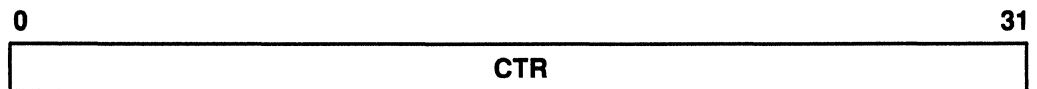
### Link Register

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 16–31 of MSR and bits 16–31 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 16–31 of SRR 1 are placed into bits 16–31 of the MSR. The MSR can also be modified by the Move to Machine State Register instruction.

<b>0</b>		<b>31</b>
<b>MSR</b>		
<b>Bit</b>	<b>Name</b>	<b>Description</b>
<b>00–15</b>		Reserved
<b>16</b>	EE	External Interrupt Enable
<b>17</b>	PR	Program State
<b>18</b>	FP	FP Available
<b>19</b>	ME	Machine Check Enable
<b>20</b>	FE	FP Exception Enable
<b>21–23</b>		Reserved
<b>24</b>	AL	Alignment Check
<b>25</b>	IP	Interrupt Prefix
<b>26</b>	IR	Instruction Relocate
<b>27</b>	DR	Data Relocate
<b>28–31</b>		Reserved.

The following are the Machine State register bit definitions:

<b>Bit(s)</b>	<b>Description</b>						
<b>0–15</b>	Reserved.						
<b>16</b>	External Interrupt Enable (EE).						
	<table> <tr> <th><b>Setting</b></th><th><b>Description</b></th></tr> <tr> <td><b>0</b></td><td>The processor is disabled against external interrupts.</td></tr> <tr> <td><b>1</b></td><td>The processor is enabled to take external interrupts.</td></tr> </table>	<b>Setting</b>	<b>Description</b>	<b>0</b>	The processor is disabled against external interrupts.	<b>1</b>	The processor is enabled to take external interrupts.
<b>Setting</b>	<b>Description</b>						
<b>0</b>	The processor is disabled against external interrupts.						
<b>1</b>	The processor is enabled to take external interrupts.						
<b>17</b>	Problem State (PR).						
	<table> <tr> <th><b>Setting</b></th><th><b>Description</b></th></tr> <tr> <td><b>0</b></td><td>The processor is privileged to execute any instruction.</td></tr> <tr> <td><b>1</b></td><td>The processor can only execute the non-privileged instructions.</td></tr> </table>	<b>Setting</b>	<b>Description</b>	<b>0</b>	The processor is privileged to execute any instruction.	<b>1</b>	The processor can only execute the non-privileged instructions.
<b>Setting</b>	<b>Description</b>						
<b>0</b>	The processor is privileged to execute any instruction.						
<b>1</b>	The processor can only execute the non-privileged instructions.						
<b>18</b>	Floating-Point (FP) Available.						
	<table> <tr> <th><b>Setting</b></th><th><b>Description</b></th></tr> <tr> <td><b>0</b></td><td>The processor cannot execute any floating-point instructions, including floating-point loads, stores and moves.</td></tr> <tr> <td><b>1</b></td><td>The processor can execute floating-point instructions.</td></tr> </table>	<b>Setting</b>	<b>Description</b>	<b>0</b>	The processor cannot execute any floating-point instructions, including floating-point loads, stores and moves.	<b>1</b>	The processor can execute floating-point instructions.
<b>Setting</b>	<b>Description</b>						
<b>0</b>	The processor cannot execute any floating-point instructions, including floating-point loads, stores and moves.						
<b>1</b>	The processor can execute floating-point instructions.						
<b>19</b>	Machine Check Enable (ME).						
	<table> <tr> <th><b>Setting</b></th><th><b>Description</b></th></tr> <tr> <td><b>0</b></td><td>Machine check interrupts are disabled.</td></tr> <tr> <td><b>1</b></td><td>Machine check interrupts are enabled.</td></tr> </table>	<b>Setting</b>	<b>Description</b>	<b>0</b>	Machine check interrupts are disabled.	<b>1</b>	Machine check interrupts are enabled.
<b>Setting</b>	<b>Description</b>						
<b>0</b>	Machine check interrupts are disabled.						
<b>1</b>	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.

**21–23** Reserved.

**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 30–31 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 MSR(AL).

**25** Interrupt Prefix (IP).

Setting	Description
0	Interrupts vectored to the effective address X'000xxxxx' where xxx is the interrupt offset.
1	Interrupts vectored to the effective address X'FFFxxxxx' 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).

Setting	Description
0	Data address translation is off.
1	Data address translation is on.



## Branch Instructions

The instruction execution sequence can be changed by the branch instructions. All instructions are on word boundaries. Thus, bits 30 and 31 of the generated branch target address are ignored by the processor unit in performing the branch.

Branch instructions compute their target addresses in one of four ways:

- Adding a constant to the address of the branch instruction.
- Specifying an absolute address (the BD or LI field is sign extended to 32 bits).
- Using the address contained in the Link register.
- Using the address contained in the Count register.

For the first two methods, the target addresses can be computed sufficiently ahead of the branch instruction so as to prefetch instructions along the target path. For the third and fourth methods, prefetching instructions along the branch path is also possible provided the Link register or the Count register is loaded sufficiently ahead of the branch instruction.

In the case of conditional branch instructions, instruction prefetching is done on each path of the branch.

In the various target forms, branches generally either branch only, branch and provide a return address, or branch conditionally. If the LK bit equals 1, the link register can be used to store the return address from an invoked subroutine. The return address is the instruction immediately following the branch instruction.

In the branch conditional instructions, the BO field combines different types of branches into one instruction. The BO field specifies how the branch is affected by or affects the Condition register and the Count register. The encoding for the BO field is described as follows:

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



## Branch (I-Form)

0	6	27	31
18	LI	AA	LK

b target address (AA = 0, LK = 0)

ba target address (AA = 1, LK = 1)

bl target address (AA = 0, LK = 0)

bla target address (AA = 1, LK = 1)

If AA equals 0, the branch target address is the sum of LI || b'00' sign extended and the address of this instruction.

If AA equals 1, the branch target address is the value, LI || b'00' sign extended.

If LK equals 1, the effective address of the instruction following the branch instruction is placed into the Link register.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

## Branch Conditional (B-Form)

0	6	11	16	27	31
16	BO	BI	BD	AA	LK

bc BO, BI, target address (AA = 0, LK = 0)

bca BO, BI, target address (AA = 1, LK = 1)

bcl BO, BI, target address (AA = 0, LK = 0)

bcla BO, BI, target address (AA = 1, LK = 1)

The BI field specifies the Condition register bit used as the condition of the branch. The BO field is used as described in "Branch Instructions" on page 2-20 .

If AA equals 0, the branch target address is the sum of BD || b'00' sign extended and the address of this instruction.

If AA equals 1, the branch target address is the value, BD || b'00' sign extended.

If LK equals 1, the effective address of the instruction following the branch instruction is placed into the Link register.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None



## Branch Conditional Register (XL-Form)

0	6	11	16	21	31
19	BO	BI	///	16	LK

bcr BO, BI (LK = 0)  
 bcrl BO, BI (LK = 1)

The BI field specifies the Condition register bit used as the condition of the branch. The BO field is used as described in "Branch Instructions" on page 2-20 and the branch target address is LR (0-29) || b'00'.

If LK = 1, the effective address of the instruction following the branch instruction is placed into the Link register.

Condition Register (CR Field 0)  
 Set: None

Fixed-Point Exception Register  
 Set: None

## Branch Conditional To Count Register (XL-Form)

0	6	11	16	21	31
19	BO	BI	///	528	LK

bcc BO, BI (LK = 0)  
 bccl BO, BI (LK = 1)

The BI field specifies the Condition register bit used as the condition of the branch. The BO field is used as described in "Branch Instructions" on page 2-20 and the branch target address is CTR (0-29) || b'00'.

The decrement CTR option is not defined for this instruction and can produce an undefined branch target address.

If LK equals 1, the effective address of the instruction following the branch instruction is placed into the Link register.

Condition register (CR Field 0)  
 Set: None

Fixed-Point Exception register  
 Set: None



---

## Supervisor Linkage Instruction

The Supervisor Linkage instruction follows:

### Supervisor Call (SC–Form)

0	6	11	16	20	26	27	31
17	///	///	FL1	LEV	FL2	SA	LK
17	///	///	SV			SA	LK

svc	LEV, FL1, FL2	(SA = 0, LK = 0)
svcl	LEV, FL1, FL2	(SA = 0, LK = 0)
svca	SV	(SA = 0, LK = 0)
svcla	SV	(SA = 0, LK = 0)

An SVC–type interrupt is generated. Bits 16–31 of the SVC instruction are placed into bits 0–15 of the Count register. Bits 16–31 of the MSR are placed into bits 16–31 of the Count register. MSR bits (EE, PR, and FE) are set to 0. MSR bits (FP, ME, AL, IP, IR, and DR) are not altered. The SRRs are not affected.

If SA equals 0, instruction fetch and execution continues at one of the 128 offsets, b'1' || LEV || b'00000', to the base effective address indicated by the setting of MSR(IP). FL1 and FL2 fields could be used for passing data to the SVC routine but are ignored by the hardware.

If SA equals 1, instruction fetch and execution continues at the offset, X'1FE0', to the base effective address indicated by the setting of MSR bit (IP).

If LK equals 1, the effective address of the instruction following the SVC instruction is placed into the Link register.

Condition register (CR Field 0)  
Set: None

Fixed–Point Exception register  
Set: None

#### Notes:

1. If SA equals 0, the FL1 and FL2 fields of the SVC instruction could have possible software uses for passing parameters to the SVC routine.
2. If SA equals 1, the SV field of the SVC instruction could have possible software uses for passing parameters to the SVC routine.
3. To insure correct operation, an SVC instruction must be preceded by an unconditional branch or a condition register op without an intervening conditional branch. If a useful instruction cannot be scheduled as specified, a no–op version of the Condition Register OR instruction can be inserted.

Instruction	No–op Version
crr	BT, BA, BB
	BT=BA=BB



---

## Trap Instructions

The trap instructions test for a specified set of conditions. If any of the conditions tested by a trap instruction are met, a trap-type program interrupt occurs. If the tested conditions are not met, instruction execution continues normally.

The contents of register RA is compared with either the sign-extended SI field or with the contents of register RB, depending on the trap instruction. This comparison results in five conditions that are ANDed with the TO field. If the result is not 0, a trap-type program interrupt occurs. These conditions are:

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.

### Trap Immediate (D-Form)

0	6	11	16	31
03	TO	RA	SI	

ti TO, RA, SI

The contents of register RA is compared with the sign-extended SI field. If any corresponding bit in the TO field and its respective condition generated as a result of the compare are both on, a trap-type program interrupt is generated.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

### Trap (X-Form)

0	6	11	16	21	31
31	TO	RA	RB	4	Rc

t TO, RA, RB

The contents of register RA is compared with the contents of register RB. If any corresponding bit in the TO field and its respective condition generated as a result of the compare are both on, a trap-type program interrupt is generated.

Condition Register (CR Field 0)  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception Register  
Set: None



---

## Condition Register Field Instruction

The Condition Register Field instruction follows:

### Move Condition Register Field (XL-Form)

0	6	9	11	14	16	21	31
19	BF	//	BFA	//	///	0	Rc

mcrf      BF, BFA

The contents of the Condition register field j, where j = BFA, are copied into the CR Field i, where i = BF. All other fields remain unchanged.

If LK equals 1, the contents of the Link register is undefined.

Condition register (CR Field 0)  
Set: CR Field i, where i = BF

Fixed-Point Exception register  
Set: None

---

## Condition Register Logical Instructions

The Condition Register Logical instructions follow:

### Condition Register Equivalent (XL-Form)

0	6	11	16	21	31
19	BT	BA	BB	289	LK

creqv      BT, BA, BB

The Condition register bit specified by the BA field is XORed with the Condition register bit specified by the BB field and the complemented result is placed into the Condition register bit specified by the BT field.

If LK equals 1, the contents of the Link register is undefined.

Condition Register (CR bit i, i = BT)  
Set: CR (BT)

Fixed-Point Exception Register  
Set: None



### Condition Register XOR (XL-Form)

0	6	11	16	21	31
19	BT	BA	BB	193	LK

crxor BT, BA, BB

The Condition register bit specified by the BA field is XORed with the Condition register bit specified by the BB field and the result is placed into the Condition register bit specified by the BT field.

If LK equals 1, the contents of the Link register is undefined.

Condition register (CR bit i, i = BT)

Set: CR (BT)

Fixed-Point Exception register

Set: None

### Condition Register AND (XL-Form)

0	6	11	16	21	31
19	BT	BA	BB	257	LK

crand BT, BA, BB

The Condition register bit specified by the BA field is ANDed with the Condition register bit specified by the BB field and the result is placed into the Condition register bit specified by the BT field.

If LK equals 1, the contents of the Link register is undefined.

Condition register (CR bit i, i = BT)

Set: CR (BT)

Fixed-Point Exception register

Set: None

### Condition Register OR (XL-Form)

0	6	11	16	21	31
19	BT	BA	BB	449	LK

cror BT, BA, BB

The Condition register bit specified by the BA field is ORed with the Condition register bit specified by the BB field and the result is placed into the Condition register bit specified by the BT field.

If LK equals 1, the contents of the Link register is undefined.

Condition register (CR bit i, i = BT)

Set: CR (BT)

Fixed-Point Exception register

Set: None



### Condition Register AND With Complement (XL-Form)

0	6	11	16	21	31
19	BT	BA	BB	129	LK

crandc BT, BA, BB

The Condition register bit specified by the BA field is ANDed with the complement of the Condition register bit specified by the BB field and the result is placed into the Condition register bit specified by the BT field.

If LK equals 1, the contents of the Link register is undefined.

Condition register (CR bit i, i = BT)  
Set: CR (BT)

Fixed-Point Exception register  
Set: None

### Condition Register OR With Complement (XL-Form)

0	6	11	16	21	31
19	BT	BA	BB	417	LK

crorc BT, BA, BB

The Condition register bit specified by the BA field is ORed with the complement of the Condition register bit specified by the BB field and the result is placed into the Condition register bit specified by the BT field.

If LK equals 1, the contents of the Link register is undefined.

Condition register (CR bit i, i = BT)  
Set: CR (BT)

Fixed-Point Exception Register  
Set: None



## Condition Register NAND (XL-Form)

0	6	11	16	21	31
19	BT	BA	BB	225	LK

crnand BT, BA, BB

The Condition register bit specified by the BA field is ANDed with the Condition register bit specified by the BB field and the complemented result is placed into the Condition register bit specified by the BT field.

If LK equals 1, the contents of the Link register is undefined.

Condition register (CR bit i, i = BT)  
Set: CR (BT)

Fixed-Point Exception register  
Set: None

## Condition Register NOR (XL-Form)

0	6	11	16	21	31
19	BT	BA	BB	33	LK

crnand BT, BA, BB

The Condition register bit specified by the BA field is ORed with the Condition register bit specified by the BB field and the complemented result is placed into the Condition register bit specified by the BT field.

If LK equals 1, the contents of the Link register is undefined.

Condition register (CR bit i, i = BT)  
Set: CR (BT)

Fixed-Point Exception register  
Set: None



---

## 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 9 for an example of the general purpose registers.

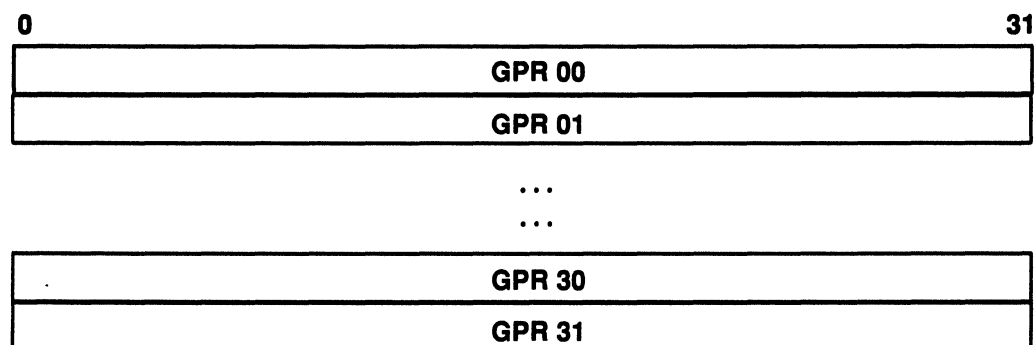


Figure 9. General Purpose Registers

### Fixed-Point Exception Register

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



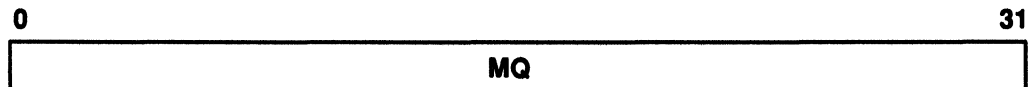
Bit	Description
0	<p>Summary Overflow (SO)</p> <p>The Summary Overflow bit is set to 1 whenever an instruction sets the Overflow bit to indicate overflow and remains set until software resets it. The SO bit is not altered by the compare instructions.</p>
1	<p>Overflow (OV)</p> <p>The Overflow bit is 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.</p>



<b>2</b>	<b>Carry (CA)</b> 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.
<b>3–15</b>	<b>Reserved</b>
<b>16–23</b>	<b>Used by the Load String and Compare Byte Indexed instruction as the byte being compared against.</b>
<b>24</b>	<b>Reserved</b>
<b>25–31</b>	<b>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.</b>

## 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.

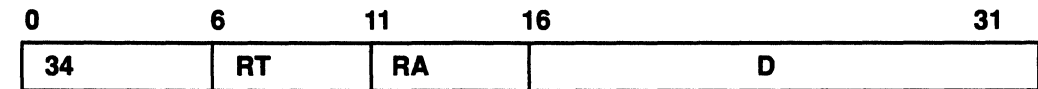




## Fixed-Point Processor Instructions

This section describes the fixed-point processor instructions used in the RISC System/6000 system. The load instructions generate the effective address (EA) as described in "Effective Address Calculation" on page 2-14. The byte, halfword, or word in memory addressed by the EA is loaded into register RT if the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt.

## Load Byte And Zero (D-Form)

**lbz**                      **RT,D(RA)**

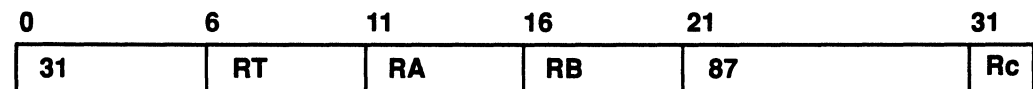
**Let the effective address (EA) be the sum  $(RA|0) + D$ .**

The byte in memory addressed by the EA is loaded into bits 24–31 of register RT. Bits 0–23 of register RT are set to 0.

**Condition register (CR Field 0)**  
**Set: None**

**Fixed-Point Exception register**  
Set: None

## Load Byte And Zero Indexed (X-Form)

**lbzx**      **RT, RA, RB**

**Let the effective address (EA) be the sum  $(RA|0) + (RB)$ .**

The byte in memory addressed by the EA is loaded into bits 24–31 of register RT. Bits 0–23 of register RT are set to 0.

**Condition register (CR Field 0)**

Set: None	(if Rc = 0)
Set: Undefined	(if Rc = 1)

**Fixed-Point Exception register**  
**Set: None**



### Load Half And Zero (D-Form)

0	6	11	16	31
40	RT	RA	D	

## lhz RT, D(RA)

Let the effective address (EA) be the sum  $(RA \ll 0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The halfword in memory addressed by the EA is loaded into bits 16–31 of register RT. Bits 0–15 of register RT are set to 0.

**Condition register (CR Field 0)**  
**Set: None**

**Fixed-Point Exception register**  
**Set: None**

## Load Half And Zero Indexed (X-Form)

0	6	11	16	21	31
31	RT	RA	RB	279	Rc

## lhzx RT, RA, RB

Let the effective address (EA) be the sum  $(RA \ll 0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

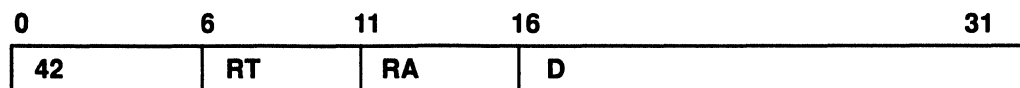
The halfword in memory addressed by the EA is loaded into bits 16–31 of register RT. Bits 0–15 of register RT are set to 0.

**Condition register (CR Field 0)**  
**Set: None** (if  $Rc = 0$ )  
**Set: Undefined** (if  $Rc = 1$ )

**Fixed-Point Exception register**  
**Set: None**



### Load Half Algebraic (D-Form)



Iha RT, D(RA)

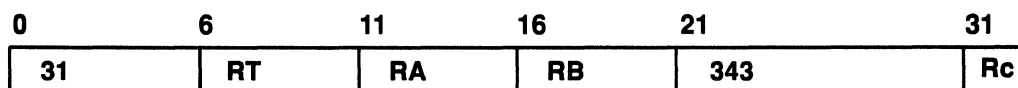
Let the effective address (EA) be the sum  $(RA[0] + D)$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The halfword in memory addressed by the EA is loaded into bits 16–31 of register RT. Bits 0–15 of register RT are filled with a copy of bit 0 of the loaded halfword.

Condition register (CR Field 0)  
Set: None

**Fixed-Point Exception register**  
**Set: None**

## Load Half Algebraic Indexed (X-Form)



lhax	RT, RA, RB
------	------------

Let the effective address (EA) be the sum (RA[0] + (RB)). If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The halfword in memory addressed by the EA is loaded into bits 16–31 of register RT. Bits 0–15 of register RT are filled with a copy of bit 0 of the loaded halfword.

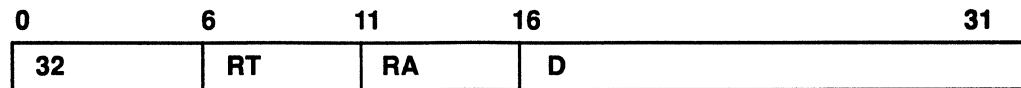
**Condition register (CR Field 0)**

Set: None	(if Rc = 0)
Set: Undefined	(if Rc = 1)

**Fixed-Point Exception register**  
**Set: None**



## Load (D-Form)



I RT, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The word in memory addressed by the EA is loaded into register RT.

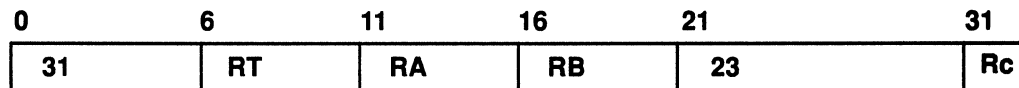
Condition register (CR Field 0)

Set: None

Fixed-Point Exception register

Set: None

## Load Indexed (X-Form)



Ix RT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The word in memory addressed by the EA is loaded into register RT.

Condition register (CR Field 0)

Set: None

(if Rc = 0)

Set: Undefined

(if Rc = 1)

Fixed-Point Exception register

Set: None



## Load Half Byte Reverse Indexed (X-Form)

0	6	11	16	21	31
31	RT	RA	RB	790	Rc

lhbrx RT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If EA addresses an I/O segment and the hardware cannot perform the access, an Alignment Interrupt is generated.

Bits 0–7 of the halfword in memory addressed by the EA are loaded into bits 24–31 of register RT. Bits 8–15 of the halfword addressed by the EA are placed into bits 16–23 of register RT. Bits 0–15 of register RT are set to 0.

Condition register (CR Field 0)

Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register

Set: None

## Load Byte Reverse Indexed (X-Form)

0	6	11	16	21	31
31	RT	RA	RB	534	Rc

lbrx RT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment and the hardware cannot perform the access, an Alignment Interrupt is generated.

Bits 0–7 of the word in memory addressed by the EA are placed into bits 24–31 of register RT. Bits 8–15 of the word addressed by the EA are placed into bits 16–23 of register RT. Bits 16–23 of the word addressed by the EA are placed into bits 8–15 of register RT. Bits 24–31 of the word addressed by the EA are placed into bits 00–07 of register RT.

Condition register (CR Field 0)

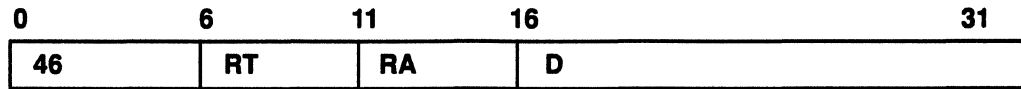
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register

Set: None



## Load Multiple (D-Form)



Im            RT, D(RA)

Let  $N$  equal  $(32 - \text{RT field})$ .

Let the effective address (EA) be the sum  $(\text{RA} \ll 0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, then an Alignment Interrupt is generated.

Starting at that effective address,  $N$  consecutive words are placed into the GPRs starting at register RT and filling all the GPRs through GPR 31.

If register RA is within the range to be loaded and  $\text{RA} \neq 0$ , data is not written into the register. The data for register RA is discarded and the operation continues normally.

Condition register (CR Field 0)  
Set: None

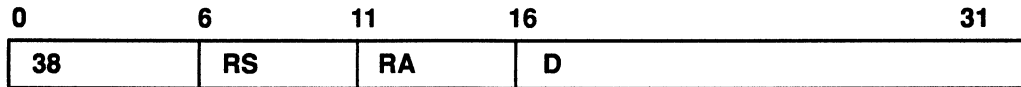
Fixed-Point Exception register  
Set: None

**Note:** A Data Storage Interrupt can interrupt this instruction. If an interrupt occurs, start the instruction from the beginning.



The store instructions generate the effective address (EA) as described in "Effective Address Calculation" on page 2-14. The contents of register RS are placed into the byte, halfword, or word in memory addressed by the EA if the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt.

### Store Byte (D-Form)



**stb**      **RS, D(RA)**

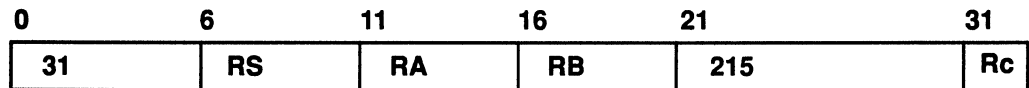
**Let the effective address (EA) be the sum  $(RA|0) + D$ .**

Bits 24–31 of register RS are placed into memory in the byte addressed by the EA. Register RS is unchanged.

**Condition register (CR Field 0)**  
**Set: None**

**Fixed-Point Exception register**  
**Set: None**

## Store Byte Indexed (X-Form)



**stbx**      RS, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ .

Bits 24–31 of register RS are placed into memory in the byte addressed by the EA. Register RS is unchanged.

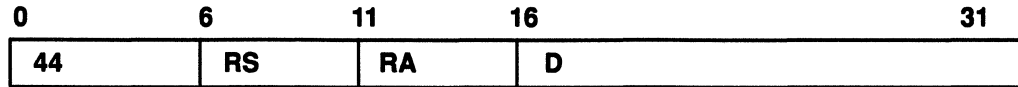
**Condition register (CR Field 0)**

Set: None	(if Rc = 0)
Set: Undefined	(if Rc = 1)

**Fixed-Point Exception register**  
**Set: None**



### Store Half (D-Form)



sth      RS, D(RA)

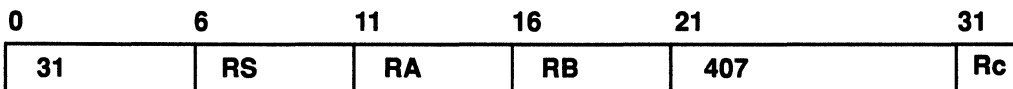
Let the effective address (EA) be the sum  $(RA \ll 0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

Bits 16–31 of register RS are placed into memory in the halfword addressed by the EA. Register RS is unchanged.

**Condition register (CR Field 0)**  
**Set: None**

**Fixed-Point Exception register**  
**Set: None**

## Store Half Indexed (X-Form)



**sthx**      **RS, RA, RB**

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

Bits 16–31 of register RS are placed into memory in the halfword addressed by the EA. Register RS is unchanged.

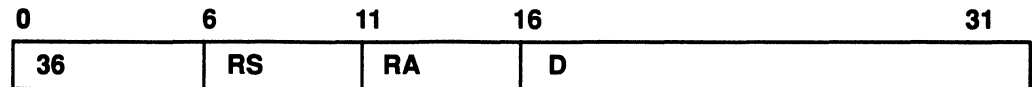
**Condition register (CR Field 0)**

Set: None	(if Rc = 0)
Set: Undefined	(if Rc = 1)

**Fixed-Point Exception register**  
**Set: None**



## Store (D-Form)



st RS, D(RA)

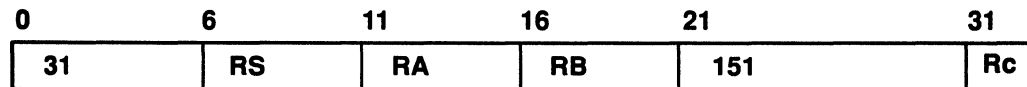
Let the effective address (EA) be the sum  $(RA[0] + D)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

Bits 0–31 of register RS are placed into memory in the word in memory addressed by the EA. Register RS is unchanged.

**Condition register (CR Field 0)**  
**Set: None**

**Fixed-Point Exception register**  
**Set: None**

## Store Indexed (X-Form)



**stx**      **RS, RA, RB**

Let the effective address (EA) be the sum  $(RA[0] + (RB))$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

Bits 0–31 of register RS are placed into memory in the word in memory addressed by the EA. Register RS is unchanged.

**Condition register (CR Field 0)**

Set: None	(if Rc = 0)
Set: Undefined	(if Rc = 1)

**Fixed-Point Exception register**  
**Set: None**



## Store Half Byte Reverse Indexed (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	918	Rc

sthbrx RS, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

Bits 24–31 of register RS are placed into memory in bits 0–7 of the halfword in memory addressed by the EA. Bits 16–23 of register RS are placed into memory in bits 8–15 of the halfword in memory addressed by the EA. Register RS is unchanged.

Condition register (CR Field 0)

Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register

Set: None

## Store Byte Reverse Indexed (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	662	Rc

stbrx RS, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

Bits 24–31 of register RS are placed into memory in bits 0–7 of the memory word addressed by the EA. Bits 16–23 of register RS are placed into memory in bits 08–15 of the memory word addressed by the EA. Bits 8–15 of register RS are placed into memory in bits 16–23 of the memory word addressed by the EA. Bits 0–7 of register RS are placed into memory in bits 24–31 of the memory word addressed by the EA. Register RS is unchanged.

Condition register (CR Field 0)

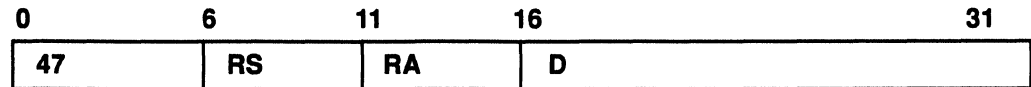
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register

Set: None



## Store Multiple (D-Form)



stm        RS, D(RA)

Let  $N$  equal (32 – RS field).

Let the effective address (EA) be the sum (RA|0) + D. If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, an Alignment Interrupt is generated.

Starting at the EA,  $N$  consecutive words are stored from register RS through register 31.

Condition register (CR Field 0)

Set: None

Fixed-Point Exception register

Set: None

**Note:** A Data Storage Interrupt can interrupt this instruction. If an interrupt occurs, start the instruction from the beginning.



## Fixed-Point Load with Update Instructions

The load with update instructions generate the effective address (EA) as described in “Effective Address Calculation” on page 2-14.

If  $RA \neq 0$ ,  $RA \neq RT$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the effective address is placed into register RA. After the update, if the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the byte, halfword, or word in memory addressed by the EA is placed into register RT.

**When RA equals RT, the register contains the data loaded from memory, not the effective address. If RA equals 0 or RA equals RT, the effective address is not saved.**

### Load Byte And Zero With Update (D-Form)

0	6	11	16	31
35	RT	RA	D	

**Ibzu**            **RT, D(RA)**

**Let the effective address (EA) be the sum  $(RA|0) + D$ .**

The byte in memory addressed by the EA is placed into bits 24–31 of register RT. Bits 0–23 of register RT are set to 0.

If  $RA \neq RT$ ,  $RA \neq 0$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

Condition register (CR Field 0)  
Set: None

**Fixed-Point Exception register**  
**Set: None**

## Load Byte And Zero With Update Indexed (X-Form)

0	6	11	16	21	31
31	RT	RA	RB	119	Rc

**Ibzux**      **RT, RA, RB**

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ .

The byte in memory addressed by the EA is placed into bits 24–31 of register RT. Bits 0–23 of register RT are set to 0.

If  $RA \neq RT$ ,  $RA \neq 0$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

**Condition register (CR Field 0)**  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

**Fixed-Point Exception register**  
**Set: None**



## Load Half And Zero With Update (D-Form)

0	6	11	16	31
41	RT	RA	D	

lhzu RT, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The halfword in memory addressed by the EA is placed into bits 16–31 of register RT. Bits 0–15 of register RT are set to 0.

If  $RA \neq RT$ ,  $RA \neq 0$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

## Load Half And Zero With Update Indexed (X-Form)

0	6	11	16	21	31
31	RT	RA	RB	311	Rc

lhzux RT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The halfword in memory addressed by the EA is placed into bits 16–31 of register RT. Bits 0–15 of register RT are set to 0.

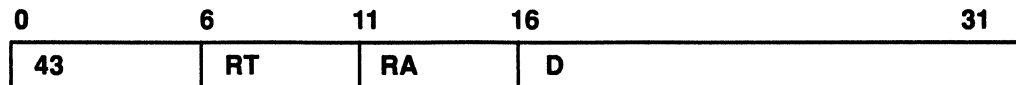
If  $RA \neq RT$ ,  $RA \neq 0$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

Condition register (CR Field 0)  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register  
Set: None



## Load Half Algebraic With Update (D-Form)



lhau RT, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

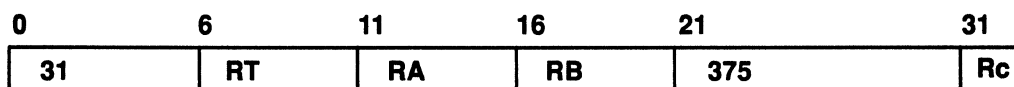
The halfword in memory addressed by the EA is placed into bits 16–31 of register RT. Bits 0–15 of register RT are filled with a copy of bit 0 of the placed halfword.

If  $RA \neq RT$ ,  $RA \neq 0$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

## Load Half Algebraic With Update Indexed (X-Form)



lhaux RT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The halfword in memory addressed by the EA is placed into bits 16–31 of register RT. Bits 0–15 of register RT are filled with a copy bit 0 of the placed halfword.

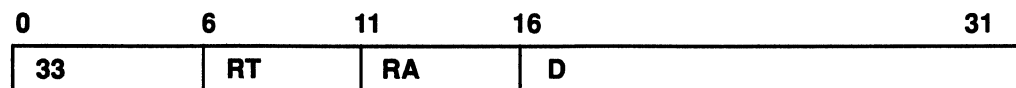
If  $RA \neq RT$ ,  $RA \neq 0$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

Condition register (CR Field 0)  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register  
Set: None



## Load With Update (D-Form)



lu RT, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The word in memory addressed by the EA is placed into register RT.

If  $RA \neq RT$ ,  $RA \neq 0$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

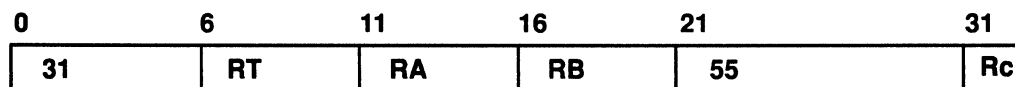
Condition register (CR Field 0)

Set: None

Fixed-Point Exception register

Set: None

## Load With Update Indexed (X-Form)



lux RT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

The word in memory addressed by the EA is placed into register RT.

If  $RA \neq RT$ ,  $RA \neq 0$ , and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

Condition register (CR Field 0)

Set: None

(if Rc = 0)

Set: Undefined

(if Rc = 1)

Fixed-Point Exception register

Set: None



## Fixed-Point Store with Update Instructions

The store with update instructions generate the effective address (EA) as described in “Effective Address Calculation” on page 2-14.

The contents of register RS are placed into memory in the byte, halfword, or word in memory addressed by the EA if the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt.

If  $RA \neq 0$  and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the effective address is placed into register RA.

In the store with update instructions, if RS equals RA, the contents of register RS are placed into memory in the byte, halfword, or word in memory addressed by the EA and the effective address is placed into register RA.

### Store Byte With Update (D-Form)

0	6	11	16	31
39	RS	RA	D	

**stbu**      **RS, D(RA)**

Let the effective address (EA) be the sum  $(RA|0) + D$ .

**Bits 24–31 of register RS are placed into memory in the memory byte addressed by the EA.**

If  $RA \neq 0$  and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the effective address is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

## Store Byte With Update Indexed (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	247	Rc

**stbux**      **RS, RA, RB**

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ .

**Bits 24–31 of register RS are placed into memory in the memory byte addressed by the EA.**

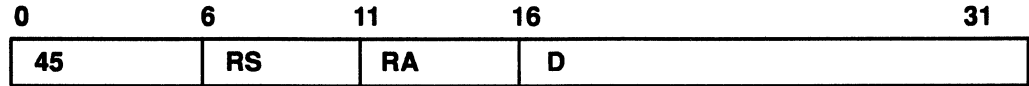
If  $RA \neq 0$  and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the effective address is placed into register RA.

**Condition register (CR Field 0)**  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

**Fixed-Point Exception register**  
**Set: None**



## Store Half With Update (D-Form)



**sth**      **RS, D(RA)**

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

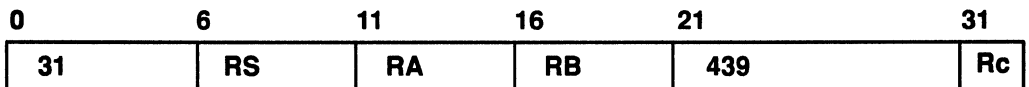
**Bits 16–31 of register RS are stored in the halfword in memory addressed by the EA.**

If  $RA \neq 0$  and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the effective address is placed into register RA.

**Condition register (CR Field 0)**  
**Set: None**

**Fixed-Point Exception register**  
**Set: None**

## Store Half With Update Indexed (X-Form)



**sthux**      RS, RA, RB

Let the effective address (EA) be the sum  $(RA[0] + (RB))$ . If alignment checking is disabled, MSR(AL) equals 0, the low-order bit of the EA is ignored. If alignment checking is enabled, MSR(AL) equals 1, and the low-order bit of the EA is not 0, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

**Bits 16–31 of register RS are stored in the halfword in memory addressed by the EA.**

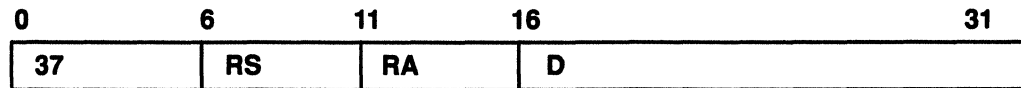
If  $RA \neq 0$  and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the effective address is placed into register RA.

**Condition register (CR Field 0)**  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

**Fixed-Point Exception register**  
**Set: None**



## Store With Update (D-Form)



stu RS, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

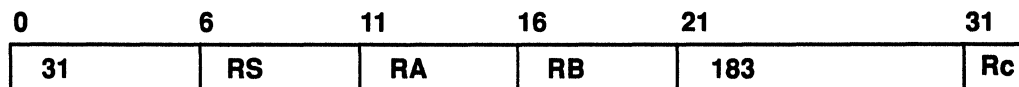
Bits 0–31 of register RS are stored in the word in memory addressed by the EA.

If  $RA \neq 0$  and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the effective address is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

## Store With Update Indexed (X-Form)



stux RS, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits of the EA are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits of the EA are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated.

Bits 0–31 of register RS are stored in the word in memory addressed by the EA.

If  $RA \neq 0$  and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the effective address is placed into register RA.

Condition register (CR Field 0)  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register  
Set: None



---

## Fixed-Point Move Assist Instructions

The string instructions allow movement of data from memory to registers or from registers to memory without concern for alignment. These instructions can be used for a short move between arbitrary memory locations or to initiate a long move between unaligned memory fields.

Load String Indexed and Store String Indexed instructions of zero length have no effect on memory, PFT entries, nor I/O if T equals 1, and do not cause data storage interrupts. Load String Indexed instructions of zero length do not alter the contents of register RT.

### Load String Indexed (X-Form)

0	6	11	16	21	31
31	RT	RA	RB	533	Rc

Isx            RT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . Let XER(25–31) contain the byte count. Let register RT be the starting register.

Let  $N$  equal XER(25–31), which is the number of bytes to be placed. Let  $NR$  equal  $\text{ceil}(N/4)$ , which is the number of registers to receive data. Starting with the leftmost byte in register RT,  $N$  consecutive bytes in memory addressed by the EA are placed into register RT through  $RT + NR - 1$ , wrapping around back through the GPR 0 if required. Bytes are always placed left to right in the register. In the case when register  $RT + NR - 1$  is only partially filled on the left, the rightmost bytes of that register are set to 0. When XER(25–31) equals 0, register RT is not altered.

Registers RA (if  $RA \neq 0$ ) and RB, if in the range to be placed, are not written into. The data that would have been written into them is discarded, and the operation continues normally. The MQ register is not affected by this operation.

Condition register (CR Field 0)

Set: None	(if Rc = 0)
Set: Undefined	(if Rc = 1)

Fixed-Point Exception register

Set: None

**Note:** A Data Storage Interrupt can interrupt this instruction. If an interrupt occurs, the instruction is restarted from the beginning.



## Load String Immediate (X-Form)

0	6	11	16	21	31
31	RT	RA	NB	597	Rc

Isl RT, RA, NB

Let the effective address (EA) be (RA|0). Let *NB* be the byte count. Let register RT be the starting register.

Let *N* equal *NB* which is the number of bytes to load. If *NB* equals 0, *N* equals 32. Let *NR* equal  $\text{ceil}(N/4)$  which is the number of registers to receive data. Starting with the leftmost byte in register RT, *N* consecutive bytes in memory addressed by the EA are placed into register RT through  $RT + NR - 1$ , wrapping around back through the GPR 0 if required. Bytes are always placed left to right in the register. In the case when register  $RT + NR - 1$  is only partially filled on the left, the rightmost bytes of that register are set to 0.

Register RA (if  $RA \neq 0$ ), if in the range to be placed, is not written into. The data that would have been written into it is discarded, and the operation continues normally. The MQ register is not affected by this operation.

Condition register (CR Field 0)

Set: None

(if Rc = 0)

Set: Undefined

(if Rc = 1)

Fixed-Point Exception register

Set: None

**Note:** A Data Storage Interrupt can interrupt this instruction. If an interrupt occurs, the instruction is restarted from the beginning.

## Load String And Compare Byte Indexed (X-Form)

0	6	11	16	21	31
31	RT	RA	RB	277	Rc

lscbx RT, RA, RB (Rc = 0)

lscbx. RT, RA, RB (Rc = 1)

Let the effective address (EA) be the sum (RA|0) + (RB). Let XER(25–31) contain the byte count. Let register RT be the starting register.

Let *N* equal XER(25–31), which is the number of bytes to be placed. Let *NR* equal  $\text{ceil}(N/4)$ , which is the number of registers to receive data.

Starting with the leftmost byte in register RT, consecutive bytes in memory addressed by the EA are placed into register RT through  $RT + NR - 1$ , wrapping around back through the GPR 0 if required, until either a byte match is found with XER16–23 or *N* bytes have been placed. If a byte match is found, that byte is also placed.

Bytes are always placed left to right in the register. In the case when a match was found before *N* bytes were placed, the contents of the rightmost bytes not placed of that register and the contents of all succeeding registers up to and including register  $RT + NR - 1$  are undefined. Also, no reference is made to memory after the matched byte is found, thus ensuring no spurious data storage interrupts are generated. In the case when a match was not found, the contents of the rightmost bytes not placed of register  $RT + NR - 1$  is undefined.

When XER(25–31) equals 0, register RT is not altered.



The count of the number of bytes placed up to and including the matched byte, if a match was found, is placed in XER(25–31).

Registers RA (if RA ≠ 0) and RB, if in the range to be placed, are not written into. The data that would have been written into them is discarded, and the operation continues normally. If the byte in XER16–23 compares with any of the four bytes that would have been placed in register RA or register RB but are being discarded for restartability, the EQ bit and the count returned in XER(25–31) are undefined.

The MQ register is not affected by this operation.

Condition register (CR Field 0)

Set: None	(if Rc = 0)
Set: LT GT EQ SO	(if Rc = 1)

Fixed–Point Exception register

Set: XER(25–31) equals number of bytes loaded

**Notes:**

1. If Rc equals 1 and XER(25–31) equals 0, CR Field 0 is undefined. If Rc equals 1 and XER(25–31) does not equal 0, CR Field 0 is set as follows:  
LT GT EQ SO equals b'00'||match||XER(SO)
2. A data storage interrupt can interrupt this instruction. If an interrupt occurs, the instruction is restarted from the beginning.
3. When the EA specifies an I/O segment, the hardware may not be able to meet the requirement that locations beyond the location containing the matching byte are not accessed. The hardware may fetch the number of bytes specified by XER(25–31) and then search for the matching byte. Accessing locations beyond the matching byte could cause spurious access violation exceptions.



## Store String Indexed (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	661	Rc

stsx RS, RA, RB

Let the effective address (EA) be the sum  $(RA[0] + (RB))$ . Let XER(25–31) contain the byte count. Let register RS be the starting register.

Let  $N$  equal XER(25–31), which is the number of bytes to store. Let  $NR$  equal  $\text{ceil}(N/4)$  which is the number of registers to store data from. Starting with the leftmost byte in register RS,  $N$  consecutive bytes are stored starting at the EA from register RS, through register  $RS + NR - 1$ .

The contents of the MQ register is undefined.

Condition register (CR Field 0)

Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register

Set: None

**Note:** A Data Storage Interrupt can interrupt this instruction. If an interrupt occurs, the instruction is restarted from the beginning.

## Store String Immediate (X-Form)

0	6	11	16	21	31
31	RS	RA	NB	725	Rc

stsi RS, RA, NB

Let the effective address (EA) be  $(RA[0])$ . Let  $NB$  be the byte count. Let register RS be the starting register.

Let  $N$  equal  $NB$ , which is the number of bytes to store. If  $NB$  equals 0,  $N$  equals 32. Let  $NR$  equal  $\text{ceil}(N/4)$  which is the number of registers to store data from. Starting with the leftmost byte in register RS,  $N$  consecutive bytes are stored starting at the address in RA from register RS, through register  $RS + NR - 1$ .

The contents of the MQ register is undefined.

Condition register (CR Field 0)

Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed-Point Exception register

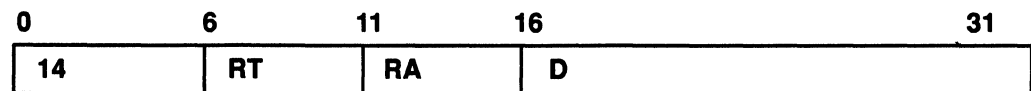
Set: None

**Note:** A Data Storage Interrupt can interrupt this instruction. If an interrupt occurs, the instruction is restarted from the beginning.



## Fixed-Point Address Computation Instructions

### Compute Address Lower (D-Form)



cal RT, D(RA)

The sum (RA[0] + D) is placed into register RT.

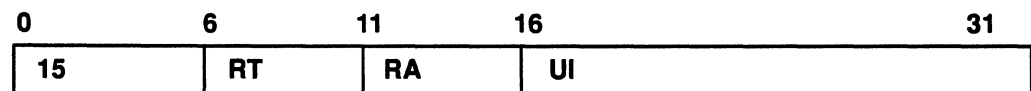
Condition register (CR Field 0)

Set: None

Fixed-Point Exception register

Set: None

### Compute Address Upper (D-Form)



cau RT, RA, UI

The sum (RA[0] + UI||X'0000') is placed into register RT.

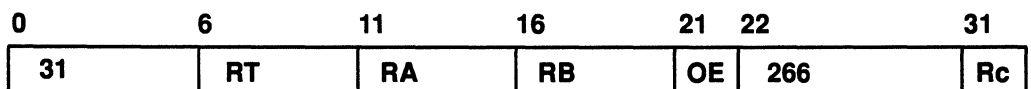
Condition register (CR Field 0)

Set: None

Fixed-Point Exception register

Set: None

### Compute Address (XO-Form)



cax RT, RA, RB (Rc = 0, OE = 0)

cax. RT, RA, RB (Rc = 1, OE = 0)

caxo RT, RA, RB (Rc = 0, OE = 1)

caxo. RT, RA, RB (Rc = 1, OE = 1)

The sum (RA) + (RB) is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None (if OE = 0)

Set: SO OV (if OE = 1)



---

## Fixed-Point Arithmetic Instructions

The arithmetic instructions treat registers as 32-bit signed integers.

The (X-Form) arithmetic instructions with Rc equals 1 and the (D-Form) arithmetic instruction, Add Immediate, set CR Field 0 by a compare of the result to zero. ai, ai., ame, aze, sfi, sfme, sfze, ae, and sfe instructions always set the CA bit to reflect the carry out of bit 0. However, the (XO-Form)s only set the CR Field 0 when Rc equals 1, and the SO and OV in the XER when OE equals 1.

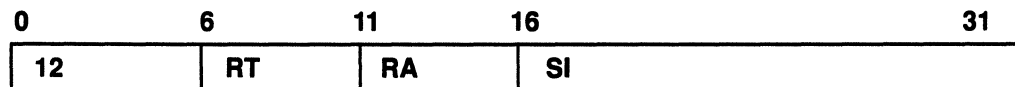
The following is the interpretation of the CR Field 0:

Bit	Name	Description
0	LT	Compares less than, negative
1	GT	Compares greater than, positive
2	EQ	Compares equal to, zero
3	SO	Summary overflow from the XER.

The following is the interpretation of the XER:

Bit	Name	Description
0	SO	Summary overflow
1	OV	Overflow
2	CA	Carry.

### Add Immediate (D-Form)



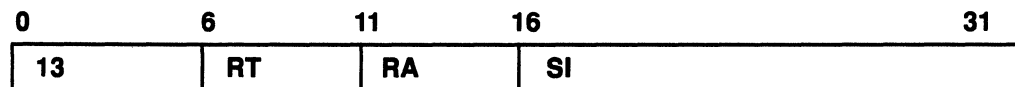
ai RT, RA, SI

The sum (RA) + SI is placed into register RT.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: CA

### Add Immediate And Record (D-Form)



ai. RT, RA, SI

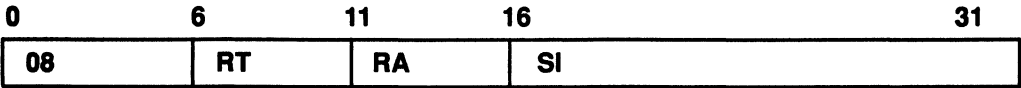
The sum (RA) + SI is placed into register RT.

Condition register (CR Field 0)  
Set: LT GT EQ SO

Fixed-Point Exception register  
Set: CA



**Subtract From Immediate (D–Form)**



sfi            RT, RA, SI

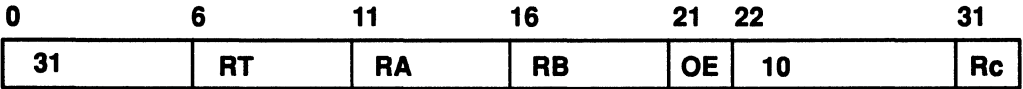
The sum  $\neg (RA) + SI + 1$  is placed into register RT.

Condition register (CR Field 0)  
Set: None

Fixed–Point Exception register  
Set: CA

**Note:** Subtract From Immediate instruction  $-1$  can be used to obtain the one's complement.

**Add (XO–Form)**



a            RT, RA, RB            (OE = 0, Rc = 0)

a.          RT, RA, RB            (OE = 0, Rc = 1)

ao          RT, RA, RB            (OE = 1, Rc = 0)

ao.        RT, RA, RB            (OE = 1, Rc = 1)

The sum  $(RA) + (RB)$  is placed into register RT.

Condition register (CR Field 0)  
Set: None                            (if Rc = 0)  
Set: LT GT EQ SO                (if Rc = 1)

Fixed–Point Exception register  
Set: CA                                (if OE = 0)  
Set: SO OV CA                      (if OE = 1)



## Subtract From (XO-Form)

0	6	11	16	21	22	31
31	RT	RA	RB	OE	8	Rc

**sf** RT, RA, RB (OE = 0, Rc = 0)

**sf.** RT, RA, RB (OE = 0, Rc = 1)

**sfo** RT, RA, RB (OE = 1, Rc = 0)

**sfo.** RT, RA, RB (OE = 1, Rc = 1)

The sum  $\neg (RA) + (RB) + 1$  is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA (if OE = 0)

Set: SO OV CA (if OE = 1)

## Add Extended (XO-Form)

0	6	11	16	21	22	31
31	RT	RA	RB	OE	138	Rc

**ae** RT, RA, RB (OE = 0, Rc = 0)

**ae.** RT, RA, RB (OE = 0, Rc = 1)

**aeo** RT, RA, RB (OE = 1, Rc = 0)

**aeo.** RT, RA, RB (OE = 1, Rc = 1)

The sum  $(RA) + (RB) + CA$  is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA (if OE = 0)

Set: SO OV CA (if OE = 1)



## Subtract From Extended (XO-Form)

0	6	11	16	21	22	31
31	RT	RA	RB	OE	136	Rc

sfe RT, RA, RB (OE = 0, Rc = 0)

sfe. RT, RA, RB (OE = 0, Rc = 1)

sfeo RT, RA, RB (OE = 1, Rc = 0)

sfeo. RT, RA, RB (OE = 1, Rc = 1)

The sum  $\neg (RA) + (RB) + CA$  is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA (if OE = 0)

Set: SO OV CA (if OE = 1)

## Add To Minus One Extended (XO-Form)

0	6	11	16	21	22	31
31	RT	RA	///	OE	234	Rc

ame RT, RA (OE = 0, Rc = 0)

ame. RT, RA (OE = 0, Rc = 1)

ameo RT, RA (OE = 1, Rc = 0)

ameo. RT, RA (OE = 1, Rc = 1)

The sum  $\neg (RA) + CA + X'FFFFFFFF'$  is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA (if OE = 0)

Set: SO OV CA (if OE = 1)



## Subtract From Minus One Extended (XO–Form)

0	6	11	16	21	22	31
31	RT	RA	///	OE	232	Rc

sfme RT, RA (OE = 0, Rc = 0)

sfme. RT, RA (OE = 0, Rc = 1)

sfmeo RT, RA (OE = 1, Rc = 0)

sfmeo. RT, RA (OE = 1, Rc = 1)

The sum  $-(RA) + CA + X'FFFFFFF'$  is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed–Point Exception register

Set: CA (if OE = 0)

Set: SO OV CA (if OE = 1)

## Add To Zero Extended (XO–Form)

0	6	11	16	21	22	31
31	RT	RA	///	OE	202	Rc

aze RT, RA (OE = 0, Rc = 0)

aze. RT, RA (OE = 0, Rc = 1)

azeo RT, RA (OE = 1, Rc = 0)

azeo. RT, RA (OE = 1, Rc = 1)

The sum  $(RA) + CA + X'00000000'$  is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed–Point Exception register

Set: CA (if OE = 0)

Set: SO OV CA (if OE = 1)



## Subtract From Zero Extended (XO–Form)

0	6	11	16	21	22	31
31	RT	RA	///	OE	200	Rc

sfze RT, RA (OE = 0, Rc = 0)

sfze. RT, RA (OE = 0, Rc = 1)

sfzeo RT, RA (OE = 1, Rc = 0)

sfzeo. RT, RA (OE = 1, Rc = 1)

The sum  $\neg (RA) + CA + X'00000000'$  is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed–Point Exception register

Set: CA (if OE = 0)

Set: SO OV CA (if OE = 1)

## Difference Or Zero Immediate (D–Form)

0	6	11	16	31
09	RT	RA	SI	

dozi RT, RA, SI

The sum  $\neg (RA) + SI + 1$  is placed into register RT. If the value in register RA is algebraically greater than the value of the SI field, register RT is set to 0.

Condition register (CR Field 0)

Set: None

Fixed–Point Exception register

Set: None

**Note:** This instruction is useful in computing the minimum and maximum of signed integers.



## Difference Or Zero (XO-Form)

0	6	11	16	21	22	31
31	RT	RA	RB	OE	264	Rc

doz RT, RA, RB (OE = 0, Rc = 0)

doz. RT, RA, RB (OE = 0, Rc = 1)

dozo RT, RA, RB (OE = 1, Rc = 0)

dozo. RT, RA, RB (OE = 1, Rc = 1)

The sum  $-(RA) + (RB) + 1$  is placed into register RT. If the value in register RA is algebraically greater than the value in register RB, register RT is set to 0. If Rc equals 1, the CR Field 0 is set to reflect the result placed in register RT (if register RT is set to 0, EQ is set to 1). If OE equals 1, the OV can only be set on positive overflows.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA (if OE = 0)

Set: SO OV (if OE = 1)

**Note:** This instruction is useful in computing the minimum and maximum of signed integers.

## Absolute (XO-Form)

0	6	11	16	21	22	31
31	RT	RA	///	OE	360	Rc

abs RT, RA (OE = 0, Rc = 0)

abs. RT, RA (OE = 0, Rc = 1)

abso RT, RA (OE = 1, Rc = 0)

abso. RT, RA (OE = 1, Rc = 1)

The absolute value  $|(RA)|$  is placed into register RT. If register RA contains the most negative number (X'80000000'), the result of the instruction is the most negative number and signals the OV bit if enabled.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA (if OE = 0)

Set: SO OV (if OE = 1)



## Negate (XO–Form)

0	6	11	16	21	22	31
31	RT	RA	///	OE	104	Rc

neg RT, RA (OE = 0, Rc = 0)

neg. RT, RA (OE = 0, Rc = 1)

nego RT, RA (OE = 1, Rc = 0)

nego. RT, RA (OE = 1, Rc = 1)

The sum  $-(RA) + 1$  is placed into register RT. If register RA contains the most negative number (X'80000000'), the result of the instruction is the most negative number and signals the OV bit if enabled.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed–Point Exception register

Set: None (if OE = 0)

Set: SO OV (if OE = 1)

## Negative Absolute (XO–Form)

0	6	11	16	21	22	31
31	RT	RA	///	OE	488	Rc

nabs RT, RA (OE = 0, Rc = 0)

nabs. RT, RA (OE = 0, Rc = 1)

nabso RT, RA (OE = 1, Rc = 0)

nabso. RT, RA (OE = 1, Rc = 1)

The negative absolute value  $-|(RA)|$  is placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed–Point Exception register

Set: None (if OE = 0)

Set: SO OV (if OE = 1)

The Negative Absolute instruction never overflows. If OE equals 1, the XER(OV) is set to 0 and XER(SO) is not changed.



## Multiply (XO-Form)

0	6	11	16	21	22	31
31	RT	RA	RB	OE	107	Rc

mul RT, RA, RB (OE = 0, Rc = 0)

mul. RT, RA, RB (OE = 0, Rc = 1)

mulo RT, RA, RB (OE = 1, Rc = 0)

mulo. RT, RA, RB (OE = 1, Rc = 1)

Bits 0–31 of the product (RA) x (RB) are placed into register RT. Bits 32–63 of the product (RA) x (RB) are placed into the MQ register.

If Rc equals 1, the LT, GT, and EQ bits reflect the result in the MQ register (the low-order 32 bits). If OE equals 1, the SO and OV bits are set to 1 if the product cannot be represented in 32 bits.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None (if OE = 0)

Set: SO OV (if OE = 1)

## Multiply Immediate (D-Form)

0	6	11	16	31
07	RT	RA	SI	

muli RT, RA, SI

Bits 32–63 of the product (RA) x SI are placed into register RT. The contents of the MQ register is undefined.

Condition register (CR Field 0)

Set: None

Fixed-Point Exception register

Set: None



## Multiply Short (XO–Form)

0	6	11	16	21	22	31
31	RT	RA	RB	OE	235	Rc

**muls**      RT, RA, RB      (OE = 0, Rc = 0)

**muls.**      RT, RA, RB      (OE = 0, Rc = 1)

**mulso**      RT, RA, RB      (OE = 1, Rc = 0)

**mulso.**      RT, RA, RB      (OE = 1, Rc = 1)

Bits 32–63 of the product (RA) x (RB) are placed into register RT. The contents of the MQ register is undefined.

If Rc equals 1, the LT, GT, and EQ bits reflect the result in register RT (the low–order 32 bits). If OE equals 1, the SO and OV bits are set to 1 if the product cannot be represented in 32 bits.

Condition register (CR Field 0)

Set: None      (if Rc = 0)

Set: LT GT EQ SO      (if Rc = 1)

Fixed–Point Exception register

Set: None      (if OE = 0)

Set: SO OV      (if OE = 1)

## Divide (XO–Form)

0	6	11	16	21	22	31
31	RT	RA	RB	OE	331	Rc

**div**      RT, RA, RB      (OE = 0, Rc = 0)

**div.**      RT, RA, RB      (OE = 0, Rc = 1)

**divo**      RT, RA, RB      (OE = 1, Rc = 0)

**divo.**      RT, RA, RB      (OE = 1, Rc = 1)

The quotient [(RA) || (MQ)] / (RB) is placed into register RT. The remainder is placed into the MQ register. The remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive. The results obey the following equation:

$$\text{dividend} = (\text{divisor} \times \text{quotient}) + \text{remainder}$$

where *dividend* is the original (RA) || (MQ), *divisor* is the original (RB), *quotient* is the final (RT), and *remainder* is the final (MQ).

If Rc equals 1, the CR bits LT, GT, and EQ reflect the remainder. If OE equals 1, the SO and OV bits are set to 1 if the quotient cannot be represented in 32 bits. For the case of  $-2^{31} + 1$ , the MQ register is set to 0 and  $-2^{31}$  is placed in register RT. For all other overflows, (MQ), (RT), and CR Field 0 (if Rc = 1) are undefined.

Condition register (CR Field 0)

Set: None      (if Rc = 0)

Set: LT GT EQ SO      (if Rc = 1)

Fixed–Point Exception register



Set: None (if OE = 0)  
Set: SO OV (if OE = 1)

## Divide Short (XO-Form)

0	6	11	16	21	22	31
31	RT	RA	RB	OE	363	Rc

divs RT, RA, RB (OE = 0, Rc = 0)

divs. RT, RA, RB (OE = 0, Rc = 1)

divso RT, RA, RB (OE = 1, Rc = 0)

divso. RT, RA, RB (OE = 1, Rc = 1)

The quotient (RA) / (RB) is placed into register RT. The remainder is placed into the MQ register. The remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive. The results obey the following equation:

$$\text{dividend} = (\text{divisor} \times \text{quotient}) + \text{remainder}$$

where *dividend* is the original (RA), *divisor* is the original (RB), *quotient* is the final (RT), and *remainder* is the final (MQ).

If Rc equals 1, the the CR bits LT, EQ and GT reflect the remainder. If OE equals 1, the SO and OV bits are set to 1 if the quotient cannot be represented in 32 bits (as is the case when the divisor is 0, or the dividend is  $-2^{31}$  and the divisor is  $-1$ ). For the case of  $-2^{31} \div -1$ , the MQ Register is set to 0 and  $-2^{31}$  is placed into register RT. For all other overflows, the (MQ), (RT), and CR Field 0 (if Rc = 1) are undefined.

Condition register (CR Field 0)

Set: None (if Rc = 0)  
Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None (if OE = 0)  
Set: SO OV (if OE = 1)



---

## Fixed–Point Compare Instructions

In compare instructions, the BF field specifies one of the CR fields that receives the result of the compare. Compare operations either logically or algebraically compare the contents of register RA with the sign extended SI field, the UI field, or the contents of register RB.

A logical compare operation is the comparison of two 32–bit unsigned integers. An algebraic compare operation is the comparison of two 32–bit signed integers. The compare operation sets one bit in the leftmost three bits of the CR field i to 1, the other two are set to 0. The XER(SO) is copied into bit 3 of CR Field i. CR Field i bits are interpreted as follows:

Bit	Name	Description
0	LT	(RA) < SI, UI, or (RB)
1	GT	(RA) > SI, UI, or (RB)
2	EQ	(RA) = SI, UI, or (RB)
3	SO	Summary Overflow from the XER

### Compare Immediate (D–Form)

0	6	9	11	16	31
11	BF	//	RA	SI	

cmpi      BF, RA, SI

The contents of register RA are compared with SI as signed integers.

Condition register

Set: CR Field i, where i = BF

Fixed–Point Exception register

Set: None

### Compare (X–Form)

0	6	9	11	16	21	31
31	BF	//	RA	RB	0	Rc

cmp      BF, RA, RB

The contents of register RA are compared with the contents of register RB as signed integers. CR Field 0 is undefined if BF ≠ 0 and Rc equals 1.

Condition register

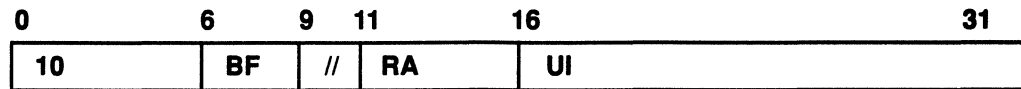
Set: CR Field i, where i = BF

Fixed–Point Exception register

Set: None



## Compare Logical Immediate (D-Form)



cmpli      BF, RA, UI

The contents of register RA are compared with X'0000' || UI as unsigned integers.

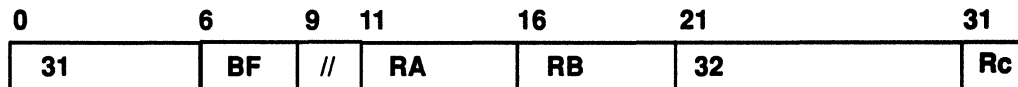
Condition register

Set: CR Field i, where i = BF

Fixed-Point Exception register

Set: None

## Compare Logical (X-Form)



cmpl      BF, RA, RB

The contents of register RA are compared with the contents of register RB as unsigned integers. CR Field 0 is undefined if BF ≠ 0 and Rc equals 1.

Condition register

Set: CR Field i, where i = BF

Fixed-Point Exception register

Set: None



---

## Fixed–Point Logical Instructions

The logical instructions perform the indicated operations by bit.

The (X–Form) logical instructions with the Rc bit set to 1 and the (D–Form) logical instructions, Add Immediate Lower and Add Immediate Upper, set bits 0–3 of the Condition register (CR Field 0) by a compare of the result to 0. The (X–Form) logical instructions with the Rc bit set to 0 and the remaining (D–Form) logical instructions do not alter the Condition register. The logical operations do not change the CA, OV and SO bits in the XER.

### AND Immediate Lower (D–Form)

0	6	11	16	31
28	RS	RA	UI	

andil. RA, RS, UI

The contents of register RS are ANDed with X'0000' || UI and the result is placed into register RA.

Condition register (CR Field 0)  
Set: LT GT EQ SO

Fixed–Point Exception register  
Set: None

### AND Immediate Upper (D–Form)

0	6	11	16	31
29	RS	RA	UI	

andiu. RA, RS, UI

The contents of register RS are ANDed with UI || X'0000' and the result is placed into register RA.

Condition register (CR Field 0)  
Set: LT GT EQ SO

Fixed–Point Exception register  
Set: None

### AND (X–Form)

0	6	11	16	21	31
31	RS	RA	RB	28	Rc

and RA, RS, RB (Rc = 0)

and. RA, RS, RB (Rc = 1)

The contents of register RS are ANDed with the contents of register RB and the result is placed into register RA.

Condition register (CR Field 0)  
Set: None (if Rc = 0)  
Set: LT GT EQ SO (if Rc = 1)



Fixed-Point Exception register  
Set: None

### OR Immediate Lower (D-Form)

0	6	11	16	31
24	RS	RA	UI	

oril RA, RS, UI

The contents of register RS are ORed with X'0000' || UI and the result is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

### OR Immediate Upper (D-Form)

0	6	11	16	31
25	RS	RA	UI	

oriu RA, RS, UI

The contents of register RS are ORed with UI || X'0000' and the result is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

### OR (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	444	Rc

or RA, RS, RB (Rc = 0)

or. RA, RS, RB (Rc = 1)

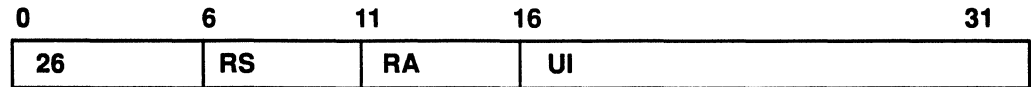
The contents of register RS are ORed with the contents of register RB and the result is placed into register RA.

Condition register (CR Field 0)  
Set: None (if Rc = 0)  
Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register  
Set: None



### XOR Immediate Lower (D-Form)



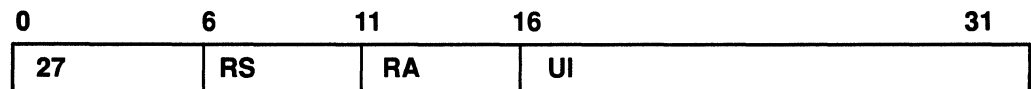
xoril      RA, RS, UI

The contents of register RS are XORed with X'0000' || UI and the result is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

### XOR Immediate Upper (D-Form)



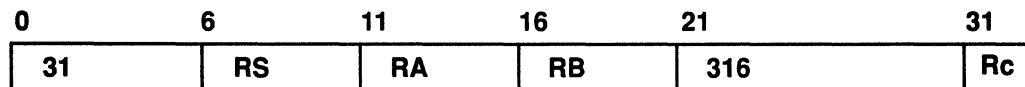
xoriu      RA, RS, UI

The contents of register RS are XORed with UI || X'0000' and the result is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed-Point Exception register  
Set: None

### XOR (X-Form)



xor      RA, RS, RB      (Rc = 0)

xor.      RA, RS, RB      (Rc = 1)

The contents of register RS are XORed with the contents of register RB and the result is placed into register RA.

Condition register (CR Field 0)  
Set: None      (if Rc = 0)  
Set: LT GT EQ SO      (if Rc = 1)

Fixed-Point Exception register  
Set: None



## Equivalent (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	284	Rc

eqv      RA, RS, RB      (Rc = 0)

eqv.     RA, RS, RB      (Rc = 1)

The contents of register RS are XORed with the contents of register RB and the complemented result is placed into register RA.

Condition register (CR Field 0)

Set: None      (if Rc = 0)

Set: LT GT EQ SO      (if Rc = 1)

Fixed-Point Exception register

Set: None

## AND With Complement (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	60	Rc

andc     RA, RS, RB      (Rc = 0)

andc.    RA, RS, RB      (Rc = 1)

The contents of register RS are ANDed with the complement of the contents of register RB and the result is placed into register RA.

Condition register (CR Field 0)

Set: None      (if Rc = 0)

Set: LT GT EQ SO      (if Rc = 1)

Fixed-Point Exception register

Set: None

## OR With Complement (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	412	Rc

orc      RA, RS, RB      (Rc = 0)

orc.     RA, RS, RB      (Rc = 1)

The contents of register RS are ORed with the complement of the contents of register RB and the result is placed into register RA.

Condition register (CR Field 0)

Set: None      (if Rc = 0)

Set: LT GT EQ SO      (if Rc = 1)

Fixed-Point Exception register

Set: None



## NOR (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	124	Rc

nor RA, RS, RB (Rc = 0)

nor. RA, RS, RB (Rc = 1)

The contents of register RS are ORed with the contents of register RB and the complemented result is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## NAND (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	476	Rc

nand RA, RS, RB (Rc = 0)

nand. RA, RS, RB (Rc = 1)

The contents of register RS are ANDed with the contents of register RB and the complemented result is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Extend Sign (X-Form)

0	6	11	16	21	31
31	RS	RA	///	922	Rc

exts RA, RS (Rc = 0)

exts. RA, RS (Rc = 1)

Bits 16–31 of register RS are placed into bits 16–31 of register RA. Bit 16 of register RS is placed into bits 0–15 of register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None



## Count Leading Zeroes (X-Form)

0	6	11	16	21	31
31	RS	RA	///	26	Rc

cntlz      RA, RS      (Rc = 0)

cntlz.      RA, RS      (Rc = 1)

The number of leading 0-bits (the number of consecutive 0-bits starting at bit 0) of the contents of register RS are placed in register RA. This number always lies between 0 and 32, inclusive.

If Rc equals 1, the LT, EQ, and GT bits are set to reflect the result. (In particular, if Rc equals 1, LT is always reset.)

Condition register (CR Field 0)

Set: None      (if Rc = 0)  
Set: LT GT EQ SO      (if Rc = 1)

Fixed-Point Exception register

Set: None



---

## Fixed-Point Rotate and Shift Instructions

The fixed-point processor performs rotate operations on data from a general purpose register and returns the result, or a portion of the result, to a general purpose register. The rotate operations move a specified number of bits left. The bits that exit from bit position 0 enter at bit position 31.

The shift instructions logically perform left and right shifts. The result of each instruction is placed into register RA under control of a generated mask.

### Fixed-Point Rotate with Mask Instructions

If Rc equals 1, the rotate instructions set bits in the CR according to the value of register RA at the completion of the instruction. The CR is set as if a compare between register RA and the value 0 had been performed. Rotate and shift operations do not change the OV and SO bits. Rotate and shift operations, except algebraic right shifts, do not change the CA bit. If Rc equals 0, the CR is left unchanged.

The result of the rotate instruction is either inserted into the register under control of the mask provided, or is ANDed with the mask before being placed into the register.

When the rotate with insert is used, the result of the rotate operation is placed into register RA under control of the provided mask. If a mask bit is 1, the associated bit of the rotated data (0 or 1) is placed into register RA; if the mask bit is 0, the associated data bit (0 or 1) from the register remains unchanged.

The rotate left instructions allow rotate right instructions to be performed (in concept) by a rotate left of  $32-N$ , where  $N$  is the number of positions to rotate right.

### Rotate Left Immediate Then Mask Insert (M-Form)

0	6	11	16	21	26	31
20	RS	RA	SH	MB	ME	Rc

rlimi      RA, RS, SH, MB, ME      (Rc = 0)

rlimi.      RA, RS, SH, MB, ME      (Rc = 1)

The contents of register RS are rotated left the number of positions specified by bits 16–20 of the instruction. The rotated data is inserted into register RA under control of the generated mask.

Condition register (CR Field 0)

Set: None      (if Rc = 0)

Set: LT GT EQ SO      (if Rc = 1)

Fixed-Point Exception register

Set: None



## Rotate Left Then Mask Insert (M–Form)

0	6	11	16	21	26	31
22	RS	RA	RB	MB	ME	Rc

rimi RA, RS, RB, MB, ME (Rc = 0)

rimi. RA, RS, RB, MB, ME (Rc = 1)

The contents of register RS are rotated left the number of positions specified by bits 27–31 of register RB. The rotated data is inserted into register RA under control of the generated mask.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed–Point Exception register

Set: None

## Rotate Left Immediate Then AND With Mask (M–Form)

0	6	11	16	21	26	31
21	RS	RA	SH	MB	ME	Rc

rlimi RA, RS, SH, MB, ME (Rc = 0)

rlimi. RA, RS, SH, MB, ME (Rc = 1)

The contents of register RS are rotated left the number of positions specified by bits 16–20 of the instruction. The rotated data is ANDed with the generated mask and the result is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed–Point Exception register

Set: None

## Rotate Left Then AND With Mask (M–Form)

0	6	11	16	21	26	31
23	RS	RA	RB	MB	ME	Rc

rlnm RA, RS, RB, MB, ME (Rc = 0)

rlnm. RA, RS, RB, MB, ME (Rc = 1)

The contents of register RS are rotated left the number of positions specified by bits 27–31 of register RB. The rotated data is ANDed with the generated mask and the result is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed–Point Exception register

Set: None



## Fixed-Point Rotate Bit Instructions

### Rotate Right And Insert Bit (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	537	Rc

rib RA, RS, RB (Rc = 0)

rib. RA, RS, RB (Rc = 1)

Bit 0 of register RS is rotated right the amount specified by bits 27–31 of register RB. The bit is then inserted into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Fixed-Point Bit Mask Instructions

### Mask Generate (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	29	Rc

mask RA, RS, RB (Rc = 0)

maskg. RA, RS, RB (Rc = 1)

Let *mstart* equal RS(27–31), specifying the starting point of a mask of ones. Let *mstop* equal RB(27–31), specifying the end point of the mask of ones.

If *mstart* < *mstop* + 1 then

MASK(*mstart*...*mstop*) equals 1s

MASK(all other bits) equals 0s

If *mstart* equals *mstop* + 1 then

MASK(0–31) equals 1s

If *mstart* > *mstop* + 1 then

MASK(*mstop* + 1...*mstart* – 1) equals 0s

MASK(all other bits) equals 1s

The MASK is then placed in register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None



## Mask Insert From Register (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	541	Rc

maskir RA, RS, RB (Rc = 0)

maskir. RA, RS, RB (Rc = 1)

Register RS is inserted into register RA under control of the mask in register RB.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Fixed-Point Shift Instructions

The instructions in this section logically perform left and right shifts.

The following process is performed when the result of a shift instruction is placed into register RA under the control of a generated mask.

When the mask bit is 1, the respective bit from either the rotated word or a word of zeros is placed into register RA. When the mask bit is 0, the respective bit from either the MQ register or a word of 32 sign bits from register RS is placed into register RA.

If the Record bit (Rc) equals 1, the shift instructions set bits in the CR according to the value of the contents of register RA at the completion of the instruction. The CR is set as if a compare between the contents of register RA and the value 0 had been performed.

If Rc equals 0, the CR is left unchanged.

## Shift Left (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	24	Rc

sl RA, RS, RB (Rc = 0)

sl. RA, RS, RB (Rc = 1)

Register RS is rotated left *N* bits where *N* is the shift amount specified in bits 27–31 of register RB.

When bit 26 of register RB is 0, a mask of 32–*N* ones followed by *N* zeros is generated.

When bit 26 of register RB is 1, a mask of all zeros is generated.

The logical AND of the rotated word and the generated mask is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None



## Shift Right (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	536	Rc

sr RA, RS, RB (Rc = 0)

sr. RA, RS, RB (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB.

When bit 26 of register RB is 0, a mask of  $N$  zeros followed by  $32-N$  ones is generated.

When bit 26 of register RB is 1, a mask of all zeros is generated.

The logical AND of the rotated word and the generated mask is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Shift Left With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	152	Rc

slq RA, RS, RB (Rc = 0)

slq. RA, RS, RB (Rc = 1)

Register RS is rotated left  $N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB. The rotated word is placed into the MQ register.

When bit 26 of register RB is 0, a mask of  $32-N$  ones followed by  $N$  zeros is generated.

When bit 26 of register RB is 1, a mask of all zeros is generated.

The logical AND of the rotated word and the generated mask is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None



## Shift Right With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	664	Rc

srq RA, RS, RB (Rc = 0)

srq. RA, RS, RB (Rc = 1)

Register RS is rotated left 32–N bits where N is the shift amount specified in bits 27–31 of register RB. The rotated word is placed into the MQ Register.

When bit 26 of register RB is 0, a mask of N zeros followed by 32–N ones is generated.

When bit 26 of register RB is 1, a mask of all zeros is generated.

The logical AND of the rotated word and the generated mask is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Shift Left Immediate With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	SH	184	Rc

sliq RA, RS, SH (Rc = 0)

sliq. RA, RS, SH (Rc = 1)

Register RS is rotated left N bits where N is the shift amount specified in bits 16–20 of the instruction. The rotated word is placed into the MQ register. A mask of 32–N ones followed by N zeros is generated. The logical AND of the rotated word and the generated mask is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None



## Shift Right Immediate With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	SH	696	Rc

sriq      RA, RS, SH      (Rc = 0)

sriq.      RA, RS, SH      (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 16–20 of the instruction. The rotated word is placed into the MQ register. A mask of  $N$  zeros followed by  $32-N$  ones is generated. The logical AND of the rotated word and the generated mask is placed into register RA.

Condition register (CR Field 0)

Set: None      (if Rc = 0)

Set: LT GT EQ SO      (if Rc = 1)

Fixed-Point Exception register

Set: None

## Shift Left Long Immediate With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	SH	248	Rc

slliq      RA, RS, SH      (Rc = 0)

slliq.      RA, RS, SH      (Rc = 1)

Register RS is rotated left  $N$  bits where  $N$  is the shift amount specified in bits 16–20 of the instruction. A mask of  $32-N$  ones followed by  $N$  zeros is generated. The rotated word is merged with the contents of the MQ register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask. The merged word is placed into register RA. The rotated word is placed into the MQ register.

Condition register (CR Field 0)

Set: None      (if Rc = 0)

Set: LT GT EQ SO      (if Rc = 1)

Fixed-Point Exception register

Set: None



## Shift Right Long Immediate With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	SH	760	Rc

srlq RA, RS, SH (Rc = 0)

srlq. RA, RS, SH (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 16–20 of the instruction. A mask of  $N$  zeros followed by  $32-N$  ones is generated. The rotated word is then merged with the contents of the MQ register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask. The merged word is placed into register RA. The rotated word is placed into the MQ register.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Shift Left Long With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	216	Rc

slq RA, RS, RB (Rc = 0)

slq. RA, RS, RB (Rc = 1)

Register RS is rotated left  $N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB.

When bit 26 of register RB is 0, a mask of  $32-N$  ones followed by  $N$  zeros is generated. The rotated word is then merged with the contents of the MQ register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask.

When bit 26 of register RB is 1, a mask of  $32-N$  zeros followed by  $N$  ones is generated. A word of zeros is then merged with the contents of the MQ register, under control of the generated mask.

The merged word is placed into register RA. The MQ register is not altered.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None



## Shift Right Long With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	728	Rc

srlq RA, RS, RB (Rc = 0)

srlq. RA, RS, RB (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB.

When bit 26 of register RB is 0, a mask of  $N$  zeros followed by  $32-N$  ones is generated. The rotated word is then merged with the contents of the MQ register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask.

When bit 26 of register RB is 1, a mask of  $N$  ones followed by  $32-N$  zeros is generated. A word of zeros is then merged with the contents of the MQ register, under control of the generated mask.

The merged word is placed into register RA. The MQ register is not altered.

Condition register (CR Field 0)

Set: None (if Rc = 0)  
Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Shift Left Extended (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	153	Rc

sle RA, RS, RB (Rc = 0)

sle. RA, RS, RB (Rc = 1)

Register RS is rotated left  $N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB. The rotated word is placed into the MQ register. A mask of  $32-N$  ones followed by  $N$  zeros is generated. The logical AND of the rotated word and the generated mask is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)  
Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None



## Shift Right Extended (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	665	Rc

sre RA, RS, RB (Rc = 0)

sre. RA, RS, RB (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB. The rotated word is placed into the MQ register. A mask of  $N$  zeros followed by  $32-N$  ones is generated. The logical AND of the rotated word and the generated mask is placed into register RA.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Shift Left Extended With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	217	Rc

sleq RA, RS, RB (Rc = 0)

sleq. RA, RS, RB (Rc = 1)

Register RS is rotated left  $N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB. A mask of  $32-N$  ones followed by  $N$  zeros is generated. The rotated word is then merged with the contents of the MQ register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask. The merged word is placed into register RA. The rotated word is placed into the MQ register.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None



## Shift Right Extended With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	729	Rc

sreq RA, RS, RB (Rc = 0)

sreq. RA, RS, RB (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB. A mask of  $N$  zeros followed by  $32-N$  ones is generated. The rotated word is then merged with the contents of the MQ register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask. The merged word is placed into register RA. The rotated word is placed into the MQ register.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

## Shift Right Algebraic Immediate (X-Form)

0	6	11	16	21	31
31	RS	RA	SH	824	Rc

srai RA, RS, SH (Rc = 0)

srai. RA, RS, SH (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 16–20 of the instruction. A mask of  $N$  zeros followed by  $32-N$  ones is generated. The rotated word is then merged with a word of 32 sign bits from the RS register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask.

The merged word is placed into register RA.

The rotated word is ANDed with the complement of the generated mask. This 32 bit result is ORed together and then ANDed with bit 0 of register RS to produce the CA bit.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: None

**Note:** All Shift Right Algebraic instructions can be used for a fast divide by  $2^N$  if followed with aze.



## Shift Right Algebraic (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	792	Rc

sra RA, RS, RB (Rc = 0)

sra. RA, RS, RB (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB. When bit 26 of register RB is 0, a mask of  $N$  zeros followed by  $32-N$  ones is generated. When bit 26 of register RB is 1, a mask of all zeros is generated. The rotated word is then merged with a word of 32 sign bits from the RS register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask.

The merged word is placed into register RA.

The rotated word is ANDed with the complement of the generated mask. This 32-bit result is ORed together and then ANDed with bit 0 of register RS to produce the CA bit.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA

**Note:** All Shift Right Algebraic instructions can be used for a fast divide by  $2^N$  if followed with a Add to Zero Extended instruction.

## Shift Right Algebraic Immediate With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	SH	952	Rc

sraiq RA, RS, SH (Rc = 0)

sraiq. RA, RS, SH (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 16–20 of the instruction. A mask of  $N$  zeros followed by  $32-N$  ones is generated. The rotated word is placed into the MQ register. The rotated word is then merged with a word of 32 sign bits from the RS register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask.

The merged word is placed into register RA.

The rotated word is ANDed with the complement of the generated mask. This 32-bit result is ORed together and then ANDed with bit 0 of register RS to produce the CA bit.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA

**Note:** All Shift Right Algebraic instructions can be used for a fast divide by  $2^N$  if followed with a Add to Zero Extended instruction.



## Shift Right Algebraic With MQ (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	920	Rc

sraq RA, RS, RB (Rc = 0)

sraq. RA, RS, RB (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB. When bit 26 of register RB is 0, a mask of  $N$  zeros followed by  $32-N$  ones is generated. When bit 26 of register RB is 1, a mask of all zeros is generated. The rotated word is placed into the MQ register. The rotated word is then merged with a word of 32 sign bits from the RS register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask.

The merged word is placed into register RA.

The rotated word is ANDed with the complement of the generated mask. This 32-bit result is ORed together and then ANDed with bit 0 of register RS to produce the CA bit.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA

**Note:** All Shift Right Algebraic instructions can be used for a fast divide by  $2^N$  if followed with a Add to Zero Extended instruction.

## Shift Right Extended Algebraic (X-Form)

0	6	11	16	21	31
31	RS	RA	RB	921	Rc

srea RA, RS, RB (Rc = 0)

srea. RA, RS, RB (Rc = 1)

Register RS is rotated left  $32-N$  bits where  $N$  is the shift amount specified in bits 27–31 of register RB. A mask of  $N$  zeros followed by  $32-N$  ones is generated. The rotated word is placed into the MQ register. The rotated word is then merged with a word of 32 sign bits from the RS register, under control of the generated mask. See "Fixed-Point Shift Instructions" on page 2-76 for information about the mask.

The merged word is placed into register RA.

The rotated word is ANDed with the complement of the generated mask. This 32-bit result is ORed together and then ANDed with bit 0 of register RS to produce the CA bit.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: LT GT EQ SO (if Rc = 1)

Fixed-Point Exception register

Set: CA



## Double-Precision Shifts

**Note:** Some of the shift instructions use the MQ register. Double-length shifting of an arbitrary pair of general purpose registers can be accomplished with a few such instructions. The shift amount is specified either as an immediate value in the instruction ( $0 \leq \text{shift amount} \leq 31$ ) or as bits 26–31 of register RB ( $0 \leq \text{shift amount} \leq 63$ ). The following examples treat registers R1 and R2 as containing a 64-bit integer, with the R1 register containing the high order part. The shift amount is given as  $n$  for the immediate shifts, and is in bits 26–31 of the R3 register for the variable shifts.

### Shift Left Double Immediate

sliq	r2, r2, n
slliq	r1, r1, n

### Shift Left Double

slq	r2, r2, r3
sllq	r1, r1, r3

### Shift Right Double Immediate

sriq	r1, r1, n
srlq	r2, r2, n

### Shift Right Double

srq	r1, r1, r3
srlq	r2, r2, r3

### Shift Right Algebraic Double Immediate

sraiq	r1, r1, n
srlq	r2, r2, n

### Shift Right Algebraic Double

cmpli	fi, r3, 32
srea	r1, r1, r3
sreq	r2, r2, r3
blt	fi, done
or	r2, r1, r1
srai	r1, r1, 31

done:



## Move To and Move From System Registers Instructions

This section defines instructions for moving data between the GPRs and the special purpose registers CTR, LR, and MQ.

## Move To Special Purpose Register (X-Form)

0	6	11	16	21	31
31	RS	SPR	///	467	Rc

**mtspr**      **SPR, RS**

The contents of register RS are placed into the special purpose register indicated by the SPR field.

SPR		Register
00000	(00)	MQ
00001	(01)	XER
01000	(08)	LR
01001	(09)	CTR

**All other combinations are reserved and do not alter any architected registers.**

### Condition register (CR Field 0)

Set: None (if  $R_c = 0$ )

Set: Undefined (if Rc = 1)

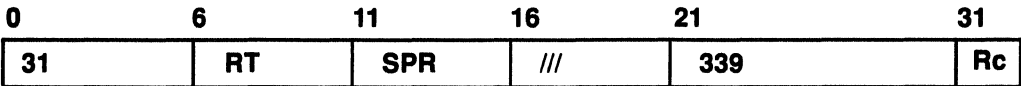
### Fixed-Point Exception register

Set: None

**Note:** Execution of this instruction, specifying SPR<sub>11</sub> set to 1 and MSR(PR) set to 1, results in a privileged instruction-type Program Interrupt.



**Move From Special Purpose Register (X-Form)**



mfspr      RT, SPR    (Rc = 0)

The contents of the special purpose register indicated by the SPR field are placed into register RT.

SPR		Register
00000	(00)	MQ
00001	(01)	XER
00100	(04)	RTCU
00101	(05)	RTCL
00110	(06)	DEC
01000	(08)	LR
01001	(09)	CTR

All other combinations are reserved and do not alter any architected registers.

Condition register (CR Field 0)

Set: None	(if Rc = 0)
Set: Undefined	(if Rc = 1)

Fixed-Point Exception register

Set: None

**Note:** Execution of this instruction, specifying SPR<sub>11</sub> set to 1 and MSR(PR) set to 1, results in a privileged instruction-type Program Interrupt.



## Move To and Move From Condition Register Instruction

This section defines instructions for moving data between the general purpose registers and the Condition register.

### Move To Condition Register Fields (XFX–Form)

0	6	11	12	20	21	31
31	RS	/	FXM	/	144	Rc

mtrcf      FXM, RS

The contents of register RS are placed into Condition register under control of the FXM field mask. FXM field mask is defined as follows:

Bit	Description
12	Bits 00–03 of CR updated
13	Bits 04–07 of CR updated
14	Bits 08–11 of CR updated
15	Bits 12–15 of CR updated
16	Bits 16–19 of CR updated
17	Bits 20–23 of CR updated
18	Bits 24–27 of CR updated
19	Bits 28–31 of CR updated.

Register RS is not changed.

Condition register (CR Field 0)

Set: See description above

(if Rc = 0)

Set: Undefined

(if Rc = 1)

Fixed–Point Exception register

Set: None

### Move To Condition Register From XER (X–Form)

0	6	9	11	16	21	31
31	BF	//	///	///	512	Rc

mcrxr      BF

The contents of XER(0–3) are copied into Condition register Field i, where i equals BF. All other fields of the Condition register remain unchanged. The XER(0–3) is reset to 0.

Condition register (CR Field 0)

Set: None

(if Rc = 0)

Set: Undefined

(if Rc equals 1 and BF ≠ 0)

Fixed–Point Exception register

Set: XER(0–3)



## Move From Condition Register (X-Form)

0	6	11	16	21	31
31	RT	///	///	19	Rc

mfcrr RT

The contents of the Condition register are placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: Undefined (if Rc = 1)

Fixed-Point Exception register

Set: None

## Move From Machine State Register Instruction

This section defines the instruction for moving data from Machine State registers.

### Move From Machine State Register (X-Form)

0	6	11	16	21	31
31	RT	///	///	83	Rc

mfmsr RT

The contents of the MSR are placed into register RT.

Condition register (CR Field 0)

Set: None (if Rc = 0)

Set: Undefined (if Rc = 1)

Fixed-Point Exception register

Set: None



---

## 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 has a dependency 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  $2^{\text{exponent}}$ . Encodings are provided in the data format to represent finite numeric values,  $\pm$  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 the "Machine State Register", MSR(FP), on page 2-18.

If the 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
  - SNaN
  - Infinity – Infinity
  - Infinity x Zero
  - Infinity ÷ Infinity
  - Zero ÷ Zero
  - 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 2-93 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 0-31. See Figure 10 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.

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.

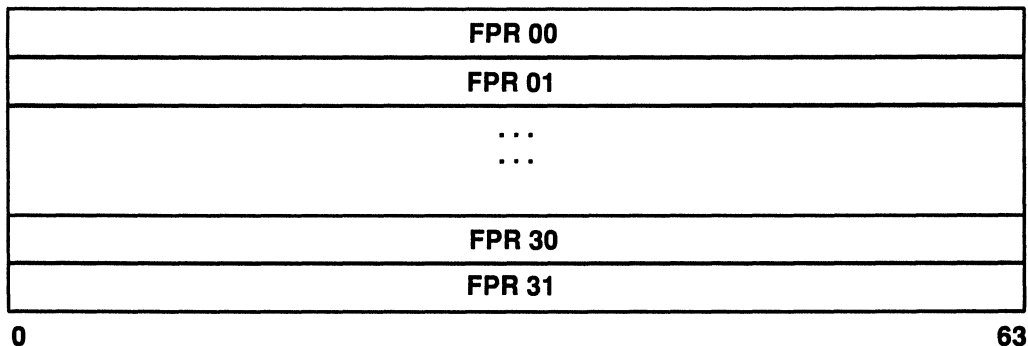


Figure 10. Floating-Point Registers



## 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 0–19 are Status bits. Bits 20–31 are Control bits.

0			31
FPSCR			
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		Reserved	
23		Reserved	
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 mtfb1 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 mtfst, mtfstf, mtfstb1, and mtfstb0 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 2-107 for information about this register.
4	Floating–Point Underflow Exception (UX). See "Underflow Exception" on page 2-109 for information about this register.
5	Floating–Point Zero Divide Exception (ZX). See "Zero Divide Exception" on page 2-106 for information about this register.
6	Floating–Point Inexact Exception (XX). See "Inexact Exception" on page 2-110 for information about this register.
7	Floating–Point Invalid Operation Exception (SNaN) (VXSNaN). See "Invalid Operation Exception" on page 2-105 for information about this register.
8	Floating–Point Invalid Operation Exception (INF – INF) (VXISI). See "Invalid Operation Exception" on page 2-105 for information about this register.
9	Floating–Point Invalid Operation Exception (INF + INF) (VXIDI). See "Invalid Operation Exception" on page 2-105 for information about this register.
10	Floating–Point Invalid Operation Exception (0 ÷ 0) (VXZDZ). See "Invalid Operation Exception" on page 2-105 for information about this register.
11	Floating–Point Invalid Operation Exception (INF × 0) (VXIMZ). See "Invalid Operation Exception" on page 2-105 for information about this register.
12	Floating–Point Invalid Operation Exception (Invalid Compare) (VXVC). See "Invalid Operation Exception" on page 2-105 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.
15–19	Floating–Point Result Flags (FPRF).



<b>Bit</b>	<b>Description</b>										
<b>15</b>	Floating-point result class descriptor (C)										
<b>16–19</b>	Floating-point condition code (FPCC).										
	<table> <tr> <th><b>Bit</b></th><th><b>Description</b></th></tr> <tr> <td><b>16</b></td><td>Floating-point less than or negative (FL or &lt;)</td></tr> <tr> <td><b>17</b></td><td>Floating-point greater than or positive (FG or &gt;)</td></tr> <tr> <td><b>18</b></td><td>Floating-point equal or zero (FE or equals)</td></tr> <tr> <td><b>19</b></td><td>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 Figure 11 on page 2-96 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.</td></tr> </table>	<b>Bit</b>	<b>Description</b>	<b>16</b>	Floating-point less than or negative (FL or <)	<b>17</b>	Floating-point greater than or positive (FG or >)	<b>18</b>	Floating-point equal or zero (FE or equals)	<b>19</b>	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 Figure 11 on page 2-96 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.
<b>Bit</b>	<b>Description</b>										
<b>16</b>	Floating-point less than or negative (FL or <)										
<b>17</b>	Floating-point greater than or positive (FG or >)										
<b>18</b>	Floating-point equal or zero (FE or equals)										
<b>19</b>	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 Figure 11 on page 2-96 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.										
<b>20–23</b>	Reserved.										
<b>24</b>	Floating-Point Invalid Operation Exception Enable (VE). See "Invalid Operation Exception" on page 2-105 for information about this register.										
<b>25</b>	Floating-Point Overflow Exception Enable (OE). See "Overflow Exception" on page 2-107 for information about this register.										
<b>26</b>	Floating-Point Underflow Exception Enable (UE). See "Underflow Exception" on page 2-109 for information about this register.										
<b>27</b>	Floating-Point Zero Divide Exception Enable (ZE). See "Zero Divide Exception" on page 2-106 for information about this register.										
<b>28</b>	Floating-Point Inexact Exception Enable (XE). See "Inexact Exception" on page 2-110 for information about this register.										
<b>29</b>	Reserved.										



**30–31** Floating–Point Rounding Control (RN). See “Rounding” on page 2-101 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 0–12) with the exception of 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: mtfstf, mtfstfi, mtfstfb0, and mcrfs.

Result Flags	Result Value Class
C < > = ?	
1 0 0 0 1	– Quiet NaN
0 1 0 0 1	– Infinity
0 1 0 0 0	– Normalized Number
1 1 0 0 0	– Denormalized Number
1 0 0 1 0	– Zero
0 0 0 1 0	+ Zero
1 0 1 0 0	+ Denormalized Number
0 0 1 0 0	+ Normalized Number
0 0 1 0 1	+ Infinity

Figure 11. Floating Point Result Flags



---

# 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 12) can be used for data in memory. The double format (See Figure 13) 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.

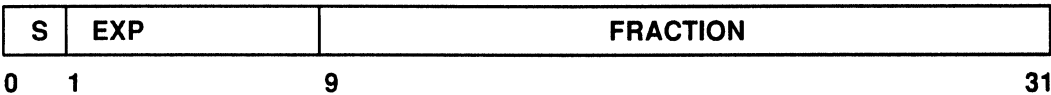


Figure 12. Floating–Point Single Format

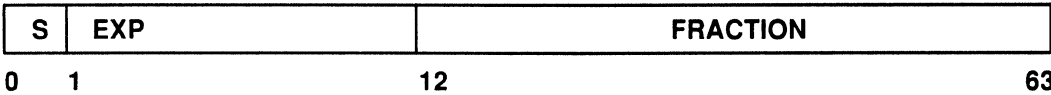


Figure 13. Floating–Point Double Format

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

- 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, and 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 14.



	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 14. 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 non-numerical 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 non-numerical 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 15 shows the relative location on the real number line for each of the defined entities.

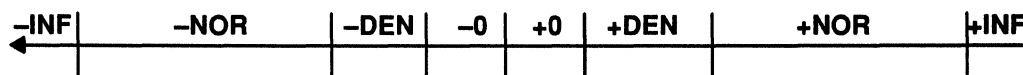


Figure 15. Approximation to Real Numbers

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 is a description of the different floating-point values defined in the architecture.

## Binary Floating-Point Numbers

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

### Normalized Numbers ( $\pm$ 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:

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



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 ( $\pm 0$ )

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 ( $\pm DEN$ )

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:

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

where  $E_{min}$  is the minimum representable exponent value (–126 for single precision, –1022 for double precision).

## Infinities ( $\pm INF$ )

Infinities are values that have the maximum biased exponent value:

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

and a zero fraction value. 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:

$$-INF < \text{every finite number} < +INF$$

Arithmetic on infinities is exact and usually does not signal an exception. Exceptions occur because of invalid operations. See “Invalid Operation Exception” on page 2-105 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 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 2-111 ) 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 2-109. 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 2-109. 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. 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 three 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:

<b>RN</b>	<b>Rounding Mode</b>
<b>00</b>	Round To Nearest
<b>01</b>	Round Towards Zero
<b>10</b>	Round Towards + Infinity
<b>11</b>	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 16 shows the relation of  $Z$ ,  $Z1$ , and  $Z2$ .



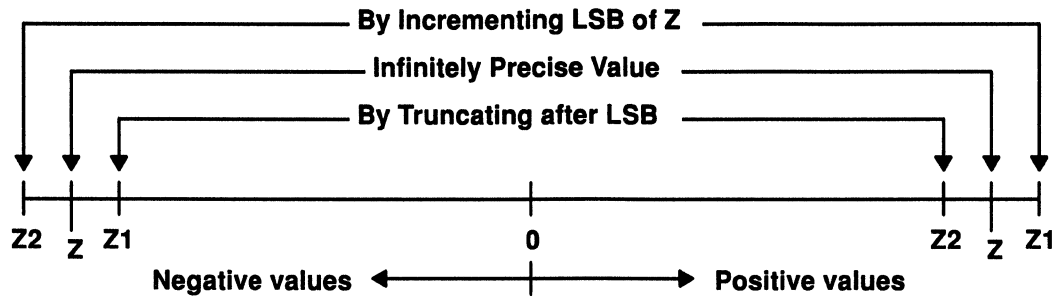


Figure 16. Selection of Z1 and Z2

The following rules specify the rounding in the four modes:

<b>Round To Nearest</b>	Choose the best approximation of Z1 or Z2. In case of a tie, choose the one that is even (least significant bit 0).
<b>Round Toward Zero</b>	Choose the smaller in magnitude (Z1 or Z2).
<b>Round Toward +Infinity</b>	Choose Z1.
<b>Round Toward -Infinity</b>	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 2-111 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 transitions 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 either requires a programmed test or enabling of program interrupts to be generated on enabled floating-point exceptions. For the programmed test to uniquely detect all exceptions that occur



precisely, each instruction that can cause a floating-point exception should be followed by a test of the FPSCR Floating-Point Exception bit. Detection of an exception can cause a software branch to an exception-handling routine. For program Interrupt detection, MSR(FE) must first be turned on, and any floating-point exception desired to be interrupted on must have its respective enable turned on.

**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 and 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 mtfstf, mtfstfi, mtfstb1, mtfstb0, or mtrcfs 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 “Floating Round to Single Precision” instruction on page 2-128 for exceptions and handling of exceptions for single-precision floating-point arithmetic.



## Invalid Operation Exception

### Definition

An Invalid Operation Exception occurs whenever 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 ( $\text{INF} - \text{INF}$ )
- Multiplication of zero by infinity ( $\text{INF} \times 0$ )
- Division of zero by zero ( $0 \div 0$ )
- Division of infinity by infinity ( $\text{INF} \div \text{INF}$ )
- Ordered comparison involving a NaN (NaN Compare).

### 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 $\text{INF} - \text{INF}$ )
FPSCR(VXIDI)	(if $\text{INF} \div \text{INF}$ )
FPSCR(VXZDZ)	(if $0 \div 0$ )
FPSCR(VXIMZ)	(if $\text{INF} \times 0$ )
FPSCR(VXVC)	(if NaN Compare).

3. If the operation is a compare operation, the FPCC field is set to reflect floating-point unordered.

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 $\text{INF} - \text{INF}$ )
FPSCR(VXIDI)	(if $\text{INF} \div \text{INF}$ )
FPSCR(VXZDZ)	(if $0 \div 0$ )
FPSCR(VXIMZ)	(if $\text{INF} \times 0$ )
FPSCR(VXVC)	(if NaN Compare).



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 floating-point unordered.

## 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 zero divide exception occurs, the following actions are taken: FPSCR(ZX)  $\leftarrow$  1.

1. Instruction execution is suppressed; operands are unmodified.
2. The Zero Divide Exception bit is set, FPSCR(ZX)  $\leftarrow$  1.

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

1. The Zero Divide Exception bit is set FPSCR(ZX)  $\leftarrow$  1.
2. The result is set to  $\pm$  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

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:

1. 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.
2. 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.
3. The magnitude of the double precision result of the arithmetic operation is less than  $2^{128}$  before rounding.
4. 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 above, the programmer decides that the error will cause significant problems for his application, either of the following methods may be used to avoid the error.

- Insure 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 may be dependent on a arithmetic operation that precedes it by less than three floating-point instructions.

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



## 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  $\text{FPSCR}(\text{OX}) \leftarrow 1$ .
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  $\text{FPSCR}(\text{OX}) \leftarrow 1$ .
2. The Inexact Exception bit is set  $\text{FPSCR}(\text{XX}) \leftarrow 1$ .
3. The result is determined by the rounding mode, FPSCR(RN), and the sign of the intermediate result as follows: for negative overflows, store  $-\text{Infinity}$ ; and, for positive overflows, store the format's largest finite number.
  - a. Round To Nearest : Store  $\pm \text{Infinity}$ , 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  $+\text{infinity}$ .
  - d. Round To  $-\text{Infinity}$ : For negative overflows, store  $-\text{infinity}$  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.



## 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 2-100 and “Rounding” on page 2-101 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  $\text{FPSCR}(\text{UX}) \leftarrow 1$ .
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  $\text{FPSCR}(\text{UX}) \leftarrow 1$ .
2. The FPRF field in the FPSCR is set to indicate the class and sign of the result ( $\pm$  Denormalized Number or  $\pm$  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:

1. The rounded result differs from the intermediate result assuming the intermediate result exponent range and precision to be unbounded.
2. 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  $\text{FPSCR}(\text{XX}) \leftarrow 1$ .
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, the floating–point instructions can be executed. Otherwise the FPP is unavailable, 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 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 the following cases when the corresponding exception enable bits 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 17 shows the format of the accumulator.

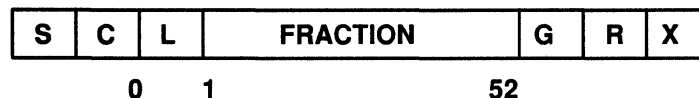


Figure 17. IEEE Execution Model

The S bit is the Sign bit.

The C bit is the Carry bit that captures the carry out of the significand.

The L bit is the Leading Unit bit of the significand that receives the implicit bit from the operands.

The FRACTION field is a 52–bit field which 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 18 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).

<b>G R X</b>	<b>Interpretation</b>
<b>0 0 0</b>	<b>IR is exact</b>
<b>0 0 0</b> <b>0 1 0</b> <b>0 1 1</b>	<b>IR closer to NL</b>
<b>1 0 0</b>	<b>IR midway between NL &amp; NH</b>
<b>1 0 1</b> <b>1 1 0</b> <b>1 1 1</b>	<b>IR closer to NH</b>

Figure 18. 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 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 into 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:

<b>RN</b>	<b>Rounding Mode</b>
<b>00</b>	<b>Round To Nearest</b>
<b>01</b>	<b>Round Toward Zero</b>
<b>10</b>	<b>Round Toward + Infinity</b>
<b>11</b>	<b>Round Toward – Infinity</b>

For rounding, the conceptual Guard, Round, and Sticky bits are defined in terms of accumulator bits. Figure 19 refers to the bit positions of Guard, Round, and Sticky for double and single-precision FP numbers.

<b>Format</b>	<b>Guard</b>	<b>Round</b>	<b>Sticky</b>
<b>Double</b> <b>Single</b>	<b>G bit</b> <b>24</b>	<b>R bit</b> <b>25</b>	<b>X bit</b> <b>26–52 G,R,X</b>

Figure 19. Location of the Guard, Round, and Sticky Bits







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

---

## Floating-Point Processor Instructions

Arithmetic operations allow implementations that range from those where the processor waits for the execution of each FPP operation to those providing for the overlapped execution of multiple operations. The instructions to load and copy the FPSCR appear to synchronize the operation of the FPP. For the copy operation, the status from all outstanding operations must be available before the contents of the FPSCR is transferred to the RT register. When the FPSCR is loaded, the status bits cannot be changed by any outstanding operations. Similarly, the execution of outstanding operations cannot be affected by new values for the FPSCR control bits. Floating-point register usage is governed by a rule of precedence which states that a register cannot be used by a given instruction until its contents reflect the results of all those instructions that precede it.

### Floating-Point Load Instructions

There are two basic forms of load instructions, single-precision and double-precision. Since the FPRs only support floating-point double-precision operands single-precision data must be converted to double-precision prior to loading into the FPR. The conversion and loading steps are as follows:

Let WORD (0-31) be the floating-point single-precision operand accessed from memory.

#### Normalized Operand

If WORD (1-8) > 0 and WORD (1-8) < 255  
FRT (0-1)  $\leftarrow$  WORD (0-1)  
FRT (2)  $\leftarrow$  WORD (1)  
FRT (3)  $\leftarrow$  WORD (1)  
FRT (4)  $\leftarrow$  WORD (1)  
FRT (5-63)  $\leftarrow$  WORD (2-31)||29 x b'0'

#### Infinity / QNaN / SNaN / Zero

If WORD (1-8) = 255 or WORD (1-31) = 0  
FRT (0-1)  $\leftarrow$  WORD (0-1)  
FRT (2)  $\leftarrow$  WORD (1)  
FRT (3)  $\leftarrow$  WORD (1)  
FRT (4)  $\leftarrow$  WORD (1)  
FRT (5-63)  $\leftarrow$  WORD (2-31)||29 x b'0'

#### Denormalized Operand

If WORD (1-8) = 0 and WORD (9-31)  $\neq$  0  
sign  $\leftarrow$  WORD (0)  
exp  $\leftarrow$  -126  
frac (0-52)  $\leftarrow$  b'0'||WORD (9-31)||29 x b'0'  
normalize the operand  
Do while frac (0) = 0  
    frac  $\leftarrow$  frac (1-52)||b'0'  
    exp  $\leftarrow$  exp - 1  
End  
FRT(0)  $\leftarrow$  sign  
FRT(1-11)  $\leftarrow$  exp + 123  
FRT(12-63)  $\leftarrow$  frac (1-52)



**Note:** The preceding description of the conversion steps are a model only. The actual implementation can vary from this but must produce results equivalent to what this model would produce.

For double-precision loads, no conversion is required as the data from memory is placed straight into the FPR.

**Note:** Recall that RA, RB, and RT denote general-purpose registers, while FRA, FRB, FRC and FRT denote floating-point registers.

### Load Floating-Point Single (D-Form)

0	6	11	16	31
48	FRT	RA	D	

lfs FRT, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The word in storage addressed by the EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision and placed into register FRT.

Condition register (CR Field 0)  
Set: None

Fixed Point Status and Control register  
Set: None

### Load Floating-Point Single Indexed (X-Form)

0	6	11	16	21	31
31	FRT	RA	RB	535	Rc

lfsx FRT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned storage access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The word in storage addressed by the EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double-precision (see "Floating-Point Load Instructions" on page 2-114) and placed into register FRT.

Condition register (CR Field 0)  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed Point Status and Control register  
Set: None



## Load Floating-Point Double (D-Form)

0	6	11	16	31
50	FRT	RA	D	

lfd FRT, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the three low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the three low-order bits are not 000, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The doubleword in memory addressed by the EA is placed into register FRT.

Condition register (CR Field 0)

Set: None

Fixed Point Status and Control register

Set: None

## Load Floating-Point Double Indexed (X-Form)

0	6	11	16	21	31
31	FRT	RA	RB	599	Rc

ldfx FRT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the three low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the three low-order bits are not 000, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The doubleword in storage addressed by the EA is and placed into register FRT.

Condition register (CR Field 0)

Set: None

(if Rc = 0)

Set: Undefined

(if Rc = 1)

Fixed Point Status and Control register

Set: None



## Load Floating–Point Single With Update (D–Form)

0	6	11	16	31
49	FRT	RA	D	

lfsu      FRT, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the two low–order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low–order bits are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The word in memory addressed by the EA is interpreted as a floating–point single–precision operand. This word is converted to floating–point double–precision and placed into register FRT. See “Floating–Point Load Instructions” on page 2-114 for information about double–precision load instructions. If  $RA \neq 0$  and the memory access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

Condition register (CR Field 0)  
Set: None

Fixed Point Status and Control register  
Set: None

## Load Floating–Point Single With Update Indexed (X–Form)

0	6	11	16	21	31
31	FRT	RA	RB	567	Rc

lfsux      FRT, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, then the two low–order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low–order bits are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

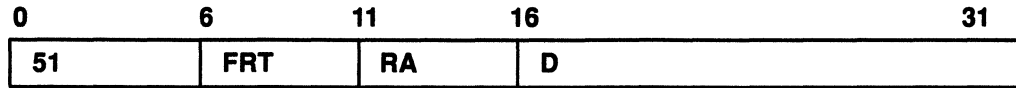
The word in memory addressed by the EA is interpreted as a floating–point single–precision operand. This word is converted to floating–point double–precision (see “Floating–Point Load Instructions” on page 2-114) and placed into register FRT. If register  $RA \neq 0$  and the storage access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

Condition register (CR Field 0)  
Set: None      (if Rc = 0)  
Set: Undefined      (if Rc = 1)

Fixed Point Status and Control register  
Set: None



## Load Floating-Point Double With Update (D-Form)



**lfdu**      **FRT, D(RA)**

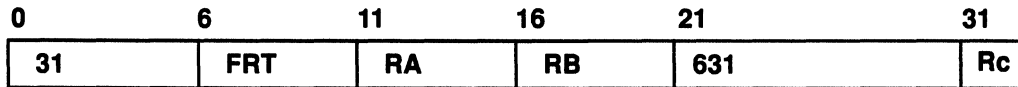
Let the effective address (EA) be the sum  $(RA[0] + D)$ . If alignment checking is disabled,  $MSR(AL)$  equals 0, the three low-order bits are ignored. If alignment checking is enabled,  $MSR(AL)$  equals 1, and the three low-order bits are not 000, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The doubleword in memory addressed by the EA is placed into register FRT. If register RA  $\neq$  0 and the storage access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

**Condition register (CR Field 0)**  
Set: None

### Fixed Point Status and Control register

## Load Floating-Point Double With Update Indexed (X-Form)



**lfdux**      **FRT, RA, RB**

Let the effective address (EA) be the sum  $(RA[0] + RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the three low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the three low-order bits are not 000, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The doubleword in memory addressed by the EA is placed into register FRT. If register RA  $\neq$  0 and the storage access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

**Condition register (CR Field 0)**  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

**Fixed Point Status and Control register**  
Set: None



## Floating-Point Store Instructions

There are two basic forms of store instructions, single-precision and double-precision. Since the FPRs only support floating-point double-precision operands, double-precision data must be converted to single-precision prior to storing operands into storage. The conversion steps follow:

Let WORD (0–31) be the word in storage written to.

### No Denormalization Required

If  $\text{FRS}(1-11) > 896$  or  $\text{FRS}(1-63) = 0$  or  
     $\text{FPSCR}(\text{UE}) = 1$   
     $\text{WORD}(0-1) \leftarrow \text{FRS}(0-1)$   
     $\text{WORD}(2-31) \leftarrow \text{FRS}(5-34)$

### Denormalized Operand

If  $\text{FRS}(1-11) \leq 896$  and  $\text{FPSCR}(\text{UE}) = 0$   
     $\text{sign} \leftarrow \text{FRS}(0)$   
     $\text{exp} \leftarrow \text{FRS}(1-11) - 1023$   
     $\text{frac} \leftarrow '1' || \text{FRS}(12-63)$   
    Denormalize the operand  
        Do while  $\text{exp} < -126$   
             $\text{frac} \leftarrow '0' || \text{frac}(0-62)$   
             $\text{exp} \leftarrow \text{exp} + 1$   
        End  
     $\text{WORD}(0) \leftarrow \text{sign}$   
     $\text{WORD}(1-8) \leftarrow \text{x'00'}$   
     $\text{WORD}(9-31) \leftarrow \text{frac}(1-23)$

#### Notes:

1. The preceding description of the conversion steps are a model only. The actual implementation can vary from this but must produce results equivalent to what this model would produce.
2. Recall that RA, RB, and RT denote general-purpose registers, while FRA, FRB, FRC, and FRT denote floating-point registers.



## Store Floating-Point Single (D-Form)

0	6	11	16	31
52	FRS	RA	D	

stfs FRS, D(RA)

Let the effective address (EA) be the sum  $(RA|0) + D$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The contents of register FRS is converted to single-precision and stored into the word in memory addressed by the EA.

Condition register (CR Field 0)  
Set: None

Fixed Point Status and Control register  
Set: None

## Store Floating-Point Single Indexed (X-Form)

0	6	11	16	21	31
31	FRS	RA	RB	663	Rc

stfsx FRS, RA, RB

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

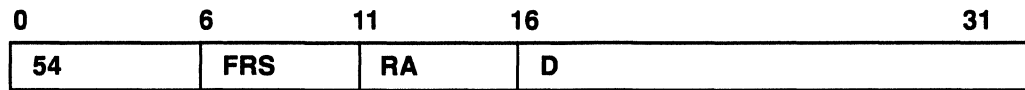
The contents of register FRS is converted to single-precision (see "Floating Point Store Instructions" on page 2-119) and stored into the word in memory addressed by the EA.

Condition register (CR Field 0)  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

Fixed Point Status and Control register  
Set: None



## Store Floating-Point Double (D-Form)



**stfd**      **FRS, D(RA)**

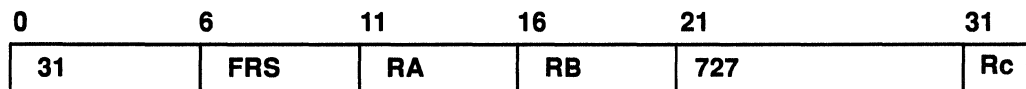
Let the effective address (EA) be the sum  $(RA[0] + D)$ . If alignment checking is disabled,  $MSR(AL)$  equals 0, the three low-order bits are ignored. If alignment checking is enabled,  $MSR(AL)$  equals 1, and the three low-order bits are not 000, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The contents of register FRS is stored into the doubleword in memory addressed by the EA. Register FRT is unchanged.

**Condition register (CR Field 0)**  
**Set: None**

**Fixed Point Status and Control register**  
**Set: None**

## Store Floating-Point Double Indexed (X-Form)



**stfdx**      **FRS, RA, RB**

Let the effective address (EA) be the sum  $(RA[0] + (RB))$ . If alignment checking is disabled, MSR(AL) equals 0, the three low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the three low-order bits are not 000, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

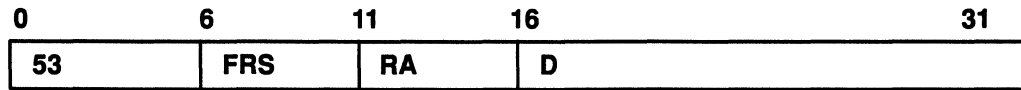
The contents of register FRS is stored into the doubleword in memory addressed by the EA. Register FRT is unchanged.

**Condition register (CR Field 0)**  
Set: None (if Rc = 0)  
Set: Undefined (if Rc = 1)

### Fixed Point Status and Control register



## Store Floating-Point Single With Update (D-Form)



**stfsu**      **FRS, D(RA)**

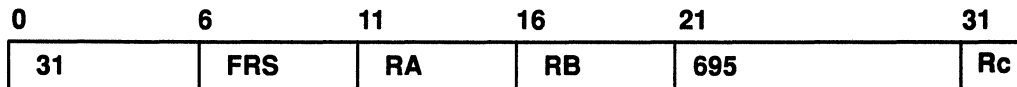
Let the effective address (EA) be the sum  $(RA[0] + D)$ . If alignment checking is disabled,  $MSR(AL)$  equals 0, the two low-order bits are ignored. If alignment checking is enabled,  $MSR(AL)$  equals 1, and the two low-order bits are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The contents of register FRS is stored into the doubleword in memory addressed by the EA. Register FRT is unchanged. If register RA  $\neq 0$  and the storage access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

**Condition register (CR Field 0)**  
**Set: None**

**Fixed Point Status and Control register**  
Set: None

## Store Floating-Point Single With Update Indexed (X-Form)



**stfsux**      **FRS, RA, RB**

Let the effective address (EA) be the sum  $(RA|0) + (RB)$ . If alignment checking is disabled, MSR(AL) equals 0, the two low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the two low-order bits are not 00, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The contents of register FRS is stored into the doubleword in memory addressed by EA. Register FRT is unchanged. If register RA  $\neq 0$  and the storage access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

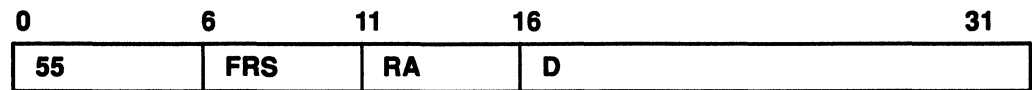
**Condition register (CR Field 0)**

Set: None	(if Rc = 0)
Set: Undefined	(if Rc = 1)

**Fixed Point Status and Control register**  
Set: None



## Store Floating-Point Double With Update (D-Form)



**stfdu**      **FRS, D(RA)**

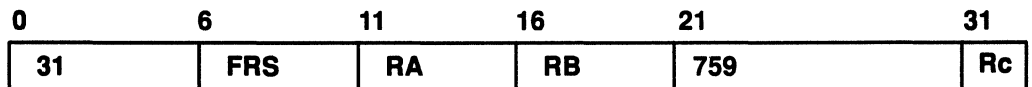
Let the effective address (EA) be the sum  $(RA[0] + D)$ . If alignment checking is disabled,  $MSR(AL)$  equals 0, the three low-order bits are ignored. If alignment checking is enabled,  $MSR(AL)$  equals 1, and the three low-order bits are not 000, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The contents of register FRS is stored into the doubleword in memory addressed by the EA. Register FRT is unchanged. If  $RA \neq 0$  and the storage access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

**Condition register (CR Field 0)**  
**Set: None**

### Fixed Point Status and Control register

## Store Floating-Point Double With Update Indexed (X-Form)

**stfdux**      **FRS, RA, RB**

Let the effective address (EA) be the sum  $(RA[0] + (RB))$ . If alignment checking is disabled, MSR(AL) equals 0, the three low-order bits are ignored. If alignment checking is enabled, MSR(AL) equals 1, and the three low-order bits are not 000, the hardware attempts to perform the unaligned memory access. If the hardware cannot perform the unaligned memory access, an Alignment Interrupt is generated. If the EA addresses an I/O segment, a Data Storage Interrupt is generated.

The contents of register FRS is stored into the doubleword in memory addressed by the EA. Register FRT is unchanged. If register RA  $\neq$  0 and the storage access does not cause an Alignment Interrupt or a Data Storage Interrupt, the EA is placed into register RA.

**Condition register (CR Field 0)**  
**Set: None** (if Rc = 0)  
**Set: Undefined** (if Rc = 1)

### Fixed Point Status and Control register



## Floating–Point Move Instructions

These instructions move data from one floating register to another with data modifications as described in each instruction description. These instructions do not modify the FPSCR and do not generate any exceptions.

The Rc bit in these instructions controls the loading of result status into Condition register Field F1. If Rc equals 1, the CR Field 1 is loaded, otherwise CR is unchanged.

### Floating Move Register (X–Form)

0	6	11	16	21	31
63	FRT	///	FRB	72	Rc

fmr FRT, FRB (Rc = 0)

fmr. FRT, FRB (Rc = 1)

The contents of register FRB is placed into register FRT.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: None

### Floating Negate (X–Form)

0	6	11	16	21	31
63	FRT	///	FRB	40	Rc

fneg FRT, FRB (Rc = 0)

fneg. FRT, FRB (Rc = 1)

The contents of register FRB with bit 0 inverted is placed into register FRT.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: None

### Floating Absolute Value (X–Form)

0	6	11	16	21	31
63	FRT	///	FRB	264	Rc

fabs FRT, FRB (Rc = 0)

fabs. FRT, FRB (Rc = 1)

The contents of register FRB with bit 0 set to 0 is placed into register FRT.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: None



## Floating Negative Absolute Value (X-Form)

0	6	11	16	21	31
63	FRT	///	FRB	136	Rc

fnabs      FRT, FRB                      (Rc = 0)

fnabs.     FRT, FRB                      (Rc = 1)

The contents of register FRB with bit 0 set to 1 is placed into register FRT.

Condition register (CR Field 1)

Set: None                                      (if Rc = 0)

Set: FX FEX VX OX                          (if Rc = 1)

Fixed Point Status and Control register

Set: None



### Floating Add (A-Form)

0	6	11	16	21	26	31
63	FRT	FRA	FRB	///	21	Rc

<b>fa</b>	<b>FRT, FRA, FRB</b>	<b>(Rc = 0)</b>
<b>fa.</b>	<b>FRT, FRA, FRB</b>	<b>(Rc = 1)</b>

The 64-bit double-precision floating-point operand in register FRA is added to the 64-bit double-precision floating-point operand in register FRB. The result is rounded under control of the Floating-Point Rounding Control field (RM) of the FPSCR and placed into register FRT.

**Addition of two floating-point numbers is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added algebraically to form an intermediate sum. All 53 bits in the significand as well as all three guard bits (G, R, and X) enter into the computation.**

If a carry occurs, the sum is shifted right one bit position and the exponent is increased by one. If the Leading significant bit (L) is not a 1, the result is normalized by shifting the significant left while decrementing the exponent until the Leading bit (L) is a 1. The X bit does not participate in the left shifts. Rather, zeros are shifted into the R bit from the right.

Tininess is checked before rounding. The unrounded result is then rounded using the mode specified by the RM field of the FPSCR. The rounded result is then checked for overflow and inexact exceptions.

When the sum of two operands with an opposite sign is exactly 0, 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 FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation Exceptions when the FPSCR(VE) bit equals 1.

### Condition register (CR Field 1)

Set:	None	(if Rc = 0)
Set:	FX FEX VX OX	(if Rc = 1)

### Fixed Point Status and Control register

Set: C FL FG FE FU FR FI  
OX UX XX  
VXSAN VXISI



## Floating Subtract (A-Form)

0	6	11	16	21	26	31
63	FRT	FRA	FRB	///	20	Rc

fs	FRT, FRA, FRB	(Rc = 0)
----	---------------	----------

fs. FRT, FRA, FRB (Rc = 1)

The 64-bit double-precision floating-point operand in register FRB is subtracted from the 64-bit double-precision floating-point operand in register FRA. The result is rounded under control of the Floating-Point Rounding Control field (RM) of the FPSCR and placed into register FRT.

The execution of the Floating Subtract instruction is identical to that of the Floating Add instruction, except that the contents of FRB participates in the operation with bit 0 inverted.

The FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation Exceptions when the FPSCR(VE) bit equals 1.

### Condition register (CR Field 1)

Set: None (if  $R_c = 0$ )

Set: FX FEX VX OX (if Rc = 1)

## Fixed Point Status and Control register

Set: C FL FG FE FU FR FI

OX UX XX

VXSAN VXS

## Floating Multiply (A-Form)

0	6	11	16	21	26	31
63	FRT	FRA	///	FRC	25	Rc

fs	FRT, FRA, FRC	(Rc = 0)
----	---------------	----------

fs. FRT, FRA, FRC (Rc = 1)

The 64-bit double-precision floating-point operand in register FRA is multiplied by the 64-bit double-precision floating-point operand in register FRC. The result is rounded under control of the Floating-Point Rounding Control field (RM) of the FPSCR and placed into register FRT.

Multiplication of two floating-point numbers is based on exponent addition and multiplication of the significands.

If an operand is a denormalized number, it is prenormalized before the operation is begun.

The FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation Exceptions when the FPSCR(VE) bit equals 1.

### Condition register (CR Field 1)

Set: None (if  $R_c = 0$ )

Set: FX FEX VX OX (if Rc = 1)

### Fixed Point Status and Control register

Set: C FL FG FE FU FR FI

OX UX XX

VXSNNAN VXIMZ



## Floating Divide (A-Form)

0	6	11	16	21	26	31
63	FRT	FRA	FRB	///	18	Rc

fd FRT, FRA, FRB (Rc = 0)

fd. FRT, FRA, FRB (Rc = 1)

The 64-bit double-precision floating-point operand in register FRA is divided by the 64-bit double-precision floating-point operand in register FRB. No remainder is preserved. The result is rounded under control of the Floating-Point Rounding Control field (RM) of the FPSCR and placed into register FRT.

The floating-point division operation is based on exponent subtraction and division of the two significands.

If an operand is a denormalized number, it is prenormalized before the operation is begun.

The FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation Exceptions when the FPSCR(VE) bit equals 1.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: C FL FG FE FU FR FI  
OX UX ZX XX  
VXSNaN VXIDI VXZDZ

## Floating Round To Single Precision (X-Form)

0	6	11	16	21	31
63	FRT	///	FRB	12	Rc

frsp FRT, FRB (Rc = 0)

frsp. FRT, FRB (Rc = 1)

The 64-bit double-precision floating-point operand in register FRB is rounded to single-precision using the rounding mode specified by the (RM) field of the FPSCR and placed into register FRT.

See "Floating Point Round to Single Model" on page 2-139 for a detailed description of the model for rounding a floating-point double-precision operand to floating-point single-precision.

The FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation (SNaN) when the FPSCR(VE) bit equals 1.

This instruction may produce incorrect results under limited circumstances. Refer to "Overflow Exception" on page 2-107 for directions on insuring the correct result.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: C FL FG FE FU FR FI  
OX UX XX  
VXSNaN



## Floating–Point Accumulate Instructions

These instructions combine a multiply and add operation without an intermediate rounding operation. The fraction part of the intermediate product is 106–bits wide where all 106 bits take part in the add or subtract portion of the instruction.

### Floating Multiply Add (A–Form)

0	6	11	16	21	26	31
63	FRT	FRA	FRB	FRC	29	Rc

fma FRT, FRA, FRC, FRB (Rc = 0)

fma. FRT, FRA, FRC, FRB (Rc = 1)

The operation  $(FRT) \leftarrow -[(FRA) \times (FRC)] + (FRB)$  is performed.

If an operand is a denormalized number, it is prenormalized before the operation is begun.

The 64–bit double–precision floating–point operand in register FRA is multiplied by the 64–bit double–precision floating–point operand in register FRC. The 64–bit double–precision floating–point operand in register FRB is added to this intermediate result. The result is rounded under control of the Floating–Point Rounding Control field (RM) of the FPSCR and placed into register FRT.

The FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation Exceptions when the FPSCR(VE) bit equals 1.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: C FL FG FE FU FR FI

OX UX XX

VXSNAN VXISI VXIMZ



## Floating Multiply Subtract (A-Form)

0	6	11	16	21	26	31
63	FRT	FRA	FRB	FRC	28	Rc

fms FRT, FRA, FRC, FRB (Rc = 0)

fms. FRT, FRA, FRC, FRB (Rc = 1)

The operation  $(FRT) \leftarrow -[(FRA) \times (FRC)] - (FRB)$  is performed.

If an operand is a denormalized number it is prenormalized before the operation is begun.

The 64-bit double-precision floating-point operand in register FRA is multiplied by the 64-bit double-precision floating-point operand in register FRC. The 64-bit double-precision floating-point operand in register FRB is subtracted from this intermediate result. The result is rounded under control of the Floating-Point Rounding Control field (RM) of the FPSCR and placed into register FRT.

The FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation Exceptions when the FPSCR(VE) bit equals 1.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: C FL FG FE FU FR FI

OX UX XX

VXSNAN VXISI VXIMZ



## Floating Negative Multiply Add (A-Form)

0	6	11	16	21	26	31
63	FRT	FRA	FRB	FRC	31	Rc

fnma FRT, FRA, FRC, FRB (Rc = 0)

fnma. FRT, FRA, FRC, FRB (Rc = 1)

The operation  $(FRT) \leftarrow - \{ [(FRA) \times (FRC)] + (FRB) \}$  is performed.

If an operand is a denormalized number, it is prenormalized before the operation is begun.

The 64-bit double-precision floating-point operand in register FRA is multiplied by the 64-bit double-precision floating-point operand in register FRC. The 64-bit double-precision floating-point operand in register FRB is added to this intermediate result. The result is rounded under control of the Floating-Point Rounding Control field (RM) of the FPSCR, negated, and placed into register FRT.

This instruction is identical to "Floating Multiply Add (A-Form)" on page 2-129, with the final result negated, but with the following exceptions:

- QNaNs propagate with no effect on their Sign bit.
- QNaNs generated as the result of a disabled Invalid Operation Exception have a sign bit of 0.
- SNaNs converted to QNaNs as the result of a disabled Invalid Operation Exception have no effect on its sign bit.

The FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation Exceptions when the FPSCR(VE) bit equals 1.

Condition register (CR Field 1)

Set: None (if Rc = 0)  
Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: C FL FG FE FU FR FI  
OX UX XX  
VXSNAN VXISI VXIMZ



## Floating Negative Multiply Subtract (A-Form)

0	6	11	16	21	26	31
63	FRT	FRA	FRB	FRC	30	Rc

fnms FRT, FRA, FRC, FRB (Rc = 0)

fnms. FRT, FRA, FRC, FRB (Rc = 1)

The operation  $(FRT) \leftarrow - \{ [(FRA) \times (FRC)] - (FRB) \}$  is performed.

If an operand is a denormalized number, it is prenormalized before the operation is begun.

The 64-bit double-precision floating-point operand in register FRA is multiplied by the 64-bit double-precision floating-point operand in register FRC. The 64-bit double-precision floating-point operand in register FRB is subtracted from this intermediate result. The result is rounded under control of the Floating-Point Rounding Control field (RM) of the FPSCR, negated, and placed into register FRT.

This instruction is identical to "Floating Multiply Subtract (A-Form)" on page 2-130, with the final result negated, but with the following exceptions:

- QNaNs propagate with no effect on their sign bit.
- QNaNs generated as the result of a disabled Invalid Operation Exception have a sign bit of 0.
- SNaNs converted to QNaNs as the result of a disabled Invalid Operation Exception have no effect on its sign bit.

The FPRF field of the FPSCR is set to the class and sign of the result except for Invalid Operation Exceptions when the FPSCR(VE) bit equals 1.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: C FL FG FE FU FR FI

OX UX XX

VXSNAN VXISI VXIMZ



## Floating-Point Compare Instructions

The IBM RISC System/6000 architecture provides two floating-point compare instructions for ordered and unordered compares. In the compare instructions, the BF value determines which field in the Condition register receives the result of the compare. One bit in the field is set to 1, the others are set to 0. The four bit compare result bits are interpreted as follows:

Bit 0	(FRA) < (FRB)
Bit 1	(FRA) > (FRB)
Bit 2	(FRA) = (FRB)
Bit 3	(FRA) ? (FRB) (Unordered)

### Floating Compare Unordered (X-Form)

0	6	9	11	16	21	31
63	BF	//	FRA	FRB	0	Rc

fcmphu BF, FRA, FRB

The 64-bit double-precision floating-point operand in register FRA is compared to the 64-bit double-precision floating-point operand in register FRB. The Floating-Point Condition Code field of the FPSCR is set to reflect the value of operand FRA with respect to operand FRB. The BF value determines which field in the Condition register receives the four FPCC bits.

If one of the operands is a NaN, either quiet or signaling, the FPCC is set to reflect unordered. If one of the operands is a signaling NaN, the VXSNaN is set.

Condition register [CR Field i, i = BF(6-8)]

Set: FL FG FE FU

Fixed Point Status and Control register

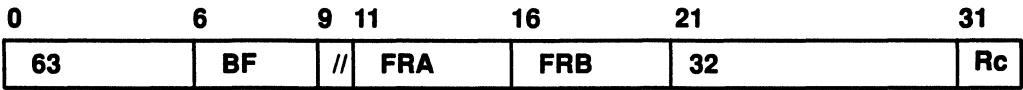
Set: FL FG FE FU

VXSNaN

**Note:** If Rc = 1, the CR Field 1 and the VXVC is undefined.



**Floating Compare Ordered (X-Form)**



fcmpo      BF, FRA, FRB

The 64-bit double-precision floating-point operand in register FRA is compared to the 64-bit double-precision floating-point operand in register FRB. The Floating-Point Condition Code field of the FPSCR is set to reflect the value of operand FRA with respect to operand FRB. The BF value determines which field in the Condition register receives the four FPCC bits.

If one of the operands is a NaN, either quiet or signaling, the FPCC is set to reflect unordered. If one of the operands is a signaling NaN, the VXSNaN is set, and if Invalid Operation is disabled (VE = 0), the VXVC is set. Otherwise, if one of the operands is a Quiet NaN, the VXVC is set.

Condition register [CR Field i, i = BF(6-8)]  
Set:    FL FG FE FU

Fixed Point Status and Control register  
Set:    FL FG FE FU  
         VXSNaN VXVC

**Note:** If Rc = 1, the CR Field 1 and the VXVC is undefined.



## Floating–Point Status and Control Register Instructions

### Move From FPSCR (X–Form)

0	6	11	16	21	31
63	FRT	///	///	583	Rc

mffs      FRT                      (Rc = 0)

mffs.    FRT                      (Rc = 1)

The contents of FPSCR is placed into bits 32–63 of floating–point register FRT.  
X'FFFFFFFF' is placed into bits 0–31 of floating point register FRT.

**Note:** This instruction loads the contents of the Floating–Point Status and Control register into an FPR, loading ones into the upper 32 bits. This makes the contents of the FPR look like a quiet NaN and is treated as one if used as an operand for any floating point–arithmetic operation.

Condition register (CR Field 1)

Set:    None                                      (if Rc = 0)

Set:    FX FEX VX OX                                      (if Rc = 1)

Fixed Point Status and Control register

Set:    None

### Move To Condition Register From FPSCR (X–Form)

0	6	9	11	14	16	21	31
63	BF	//	BFA	//	///	64	Rc

mcrfs      BF, BFA

The four bits of the Floating–Point Status and Control register, determined by the BFA field, are copied to CR Field i (i = BF). All other CR bits are unchanged.

If the field specified by the BFA contains reserved or undefined bits, 0 bits are supplied for the copy.

BFA specifies one of the 4–bit fields, 0–7, of the FPSCR.

Condition register (CR Field 1)

Set:    None                                      (if Rc = 0)

Set:    FX FEX VX OX                                      (if Rc = 1)

Fixed Point Status and Control register

Reset:FX OX                                      (BFA = 0)

UX ZX XX VXSNAN                                      (BFA = 1)

VXISI VXIDI VXZDZ VXIMZ                                      (BFA = 2)

VXVC                                      (BFA = 3)

**Note:** If Rc = 1 and the BF field ≠ 1, the CR Field 1 is undefined.



<b>0</b>	<b>6 7</b>	<b>15 16</b>	<b>21</b>	<b>31</b>
<b>63</b>	<b>/ FLM</b>	<b>/ FRB</b>	<b>711</b>	<b>Rc</b>

**FLM is a field mask, defined as follows:**

Bits 32–63 of the contents of the floating-point register FRB are placed into FPSCR under control of the field mask specified by the FLM.

Set: None (if Rc = 0)  
Set: FX FEX VX OX (if Rc = 1)

**Note:** This instruction is synchronizing within the floating-point unit and tends to hold off execution of subsequent floating-point RR operations. When specifying FPSCR 0–3, bit 3 (OX) can transition from 0 to 1. However, bit 0 (FX) is set or reset explicitly by the instruction. Also, bits 1–2 cannot be explicitly set or reset.



## Move To FPSCR Field Immediate (X-Form)

0	6	9	11	16	20	21	31
63	BF	//	///	I	/	134	Rc

mtfsfi BF, I (Rc = 0)

mtfsfi. BF, I (Rc = 1)

Bits 16–19 of the instruction are placed into the field of the FPSCR specified by the BF field. All other fields of the FPSCR are unchanged.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: Field i, where i = BF

**Note:** This instruction is synchronizing within the floating-point unit and tends to hold off execution of subsequent floating-point RR operations. When specifying FPSCR 0–3, bit 3 (OX) can transition from 0 to 1. However, bit 0 (FX) is set or reset explicitly by the instruction. Also, bits 1–2 cannot be explicitly set or reset.

## Move To FPSCR Bit 1 (X-Form)

0	6	11	16	21	31
63	BT	///	///	38	Rc

mtfsb1 BT (Rc = 0)

mtfsb1. BT (Rc = 1)

The bit specified by the BT field in FPSCR is set to 1. All other bits of the FPSCR are unchanged.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: Bit i, where i = BT

**Note:** This instruction is synchronizing within the floating-point unit and tends to hold off execution of subsequent floating-point RR operations. Also, bits 1–2 cannot be explicitly set or reset.



## Move To FPSCR Bit 0 (X-Form)

0	6	11	16	21	31
63	BT	///	///	70	Rc

mtfsb0 BT (Rc = 0)

mtfsb0. BT (Rc = 1)

The bit specified by the BT field in FPSCR is set to 0. All other bits of the FPSCR are unchanged.

Condition register (CR Field 1)

Set: None (if Rc = 0)

Set: FX FEX VX OX (if Rc = 1)

Fixed Point Status and Control register

Set: Bit i, where i = BT

**Note:** This instruction is synchronizing within the floating-point unit and tends to hold off execution of subsequent floating point RR operations. Also, bits 1–2 cannot be explicitly set or reset.



## Floating Point Round to Single Model

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

### Floating 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
  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
  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
  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
  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
```



## RISC System/6000 Instruction Set

Mnemonic	Instruction	Format	Primary Opcode	Extended Opcode
a[o][.]	Add	XO	31	10
abs[o][.]	Absolute	XO	31	360
ae[o][.]	Add Extended	XO	31	138
ai	Add Immediate	D	12	
ai.	Add Immediate And Record	D	13	
ame[o][.]	Add To Minus One Extended	XO	31	234
and[.]	AND	X	31	28
andc[.]	AND With Complement	X	31	60
andil.	AND Immediate Lower	D	28	
andiu.	AND Immediate Upper	D	29	
aze[o][.]	Add To Zero Extended	XO	31	202
b[l][a]	Branch	I	18	
bc[l][a]	Branch Conditional	B	16	
bcc[l]	Branch Conditional To Count Register	XL	19	528
bcr[l]	Branch Conditional Register	XL	19	16
cal	Compute Address Lower	D	14	
cau	Compute Address Upper	D	15	
cax[o][.]	Compute Address	XO	31	266
cmp	Compare	X	31	0
cmpi	Compare Immediate	D	11	
cmpl	Compare Logical	X	31	32
cmpi	Compare Logical Immediate	D	10	
cntlz[.]	Count Leading Zeroes	X	31	26
crand	Condition Register AND	XL	19	257
crandc	Condition Register AND With Complement	XL	19	129
creqv	Condition Register Equivalent	XL	19	289
crnand	Condition Register NAND	XL	19	225
crnor	Condition Register NOR	XL	19	33
cror	Condition Register OR	XL	19	449
crorc	Condition Register OR With Complement	XL	19	417
crxor	Condition Register XOR	XL	19	193
div[o][.]	Divide	XO	31	331
divs[o][.]	Divide Short	XO	31	363
doz[o][.]	Difference Or Zero	XO	31	264
dozi	Difference Or Zero Immediate	D	09	



<b>Mnemonic</b>	<b>Instruction</b>	<b>Format</b>	<b>Primary Opcode</b>	<b>Extended Opcode</b>
eqv[.]	Equivalent	X	31	284
exts[.]	Extend Sign	X	31	922
fa[.]	Floating Add	A	63	21
fabs[.]	Floating Absolute Value	X	63	264
fcmpo	Floating Compare Ordered	X	63	32
fcmpu	Floating Compare Unordered	X	63	0
fd[.]	Floating Divide	A	63	8
fm[.]	Floating Multiply	A	63	5
fma[.]	Floating Multiply Add	A	63	29
fmr[.]	Floating Move Register	X	63	72
fms[.]	Floating Multiply Subtract	A	63	28
fnabs[.]	Floating Negative Absolute Value	X	63	136
fneg[.]	Floating Negate	X	63	40
fnma[.]	Floating Negative Multiply Add	A	63	31
fnms[.]	Floating Negative Multiply Subtract	A	63	30
frsp[.]	Floating Round To Single Precision	X	63	12
fs[.]	Floating Subtract	A	63	20
l	Load	D	32	
lbrx	Load Byte Reverse Indexed	X	31	534
lbz	Load Byte And Zero	D	34	
lbzu	Load Byte And Zero With Update	D	35	
lbzux	Load Byte And Zero With Update Indexed	X	31	119
lbzx	Load Byte And Zero Indexed	X	31	87
lfd	Load Floating–Point Double	D	50	
lfdu	Load Floating–Point Double With Update	D	51	
lfdux	Load Floating–Point Double With Update Indexed	X	31	631
lfdx	Load Floating–Point Double Indexed	X	31	599
lfs	Load Floating–Point Single	D	48	
lfsu	Load Floating–Point Single With Update	D	49	
lfsux	Load Floating–Point Single With Update Indexed	X	31	567
lfsx	Load Floating–Point Single Indexed	X	31	535
lha	Load Half Algebraic	D	42	
lhau	Load Half Algebraic With Update	D	43	



<b>Mnemonic</b>	<b>Instruction</b>	<b>Format</b>	<b>Primary Opcode</b>	<b>Extended Opcode</b>
lhax	Load Half Algebraic With Update Indexed	X	31	375
lhax	Load Half Algebraic Indexed	X	31	343
lhbrx	Load Half Byte Reverse Indexed	X	31	790
lhz	Load Half And Zero	D	40	
lhzu	Load Half And Zero With Update	D	41	
lhzux	Load Half And Zero With Update Indexed	X	31	311
lhzx	Load Half And Zero Indexed	X	31	279
lm	Load Multiple	D	46	
lscbx[.]	Load String And Compare Byte Indexed	X	31	277
lsi	Load String Immediate	X	31	597
lsx	Load String Indexed	X	31	533
lu	Load With Update	D	33	
lux	Load With Update Indexed	X	31	55
lx	Load Indexed	X	31	23
maskg[.]	Mask Generate	X	31	29
maskir[.]	Mask Insert From Register	X	31	541
mcrf	Move Condition Register Field	XL	19	0
mcrfs	Move To Condition Register From FPSCR	X	63	64
mcrxr	Move To Condition Register From XER	X	31	512
mfcrr	Move From Condition Register	X	31	19
mffs[.]	Move From FPSCR	X	63	583
mfmsr	Move From Machine State Register	X	31	83
mfsprr	Move From Special Purpose Register	X	31	339
mtcrf	Move To Condition Register Fields	XFX	31	144
mtfsb0[.]	Move To FPSCR Bit 0	X	63	70
mtfsb1[.]	Move To FPSCR Bit 1	X	63	38
mtfsf[.]	Move To FPSCR Fields	XFL	63	711
mtfsfi[.]	Move To FPSCR Field Immediate	X	63	134
mtspr	Move To Special Purpose Register	X	31	467
mul[o][.]	Multiply	XO	31	107
muli	Multiply Immediate	D	07	
muls[o][.]	Multiply Short	XO	31	235
nabs[o][.]	Negative Absolute	XO	31	488
nand[.]	NAND	X	31	476
neg[o][.]	Negate	XO	31	104



<b>Mnemonic</b>	<b>Instruction</b>	<b>Format</b>	<b>Primary Opcode</b>	<b>Extended Opcode</b>
nor[.]	NOR	X	31	124
or[.]	OR	X	31	444
orc[.]	OR With Complement	X	31	412
oril	OR Immediate Lower	D	24	
oriu	OR Immediate Upper	D	25	
rlimi[.]	Rotate Left Immediate Then Mask Insert	M	20	
rlinm[.]	Rotate Left Immediate Then AND With Mask	M	21	
rlmi[.]	Rotate Left Then Mask Insert	M	22	
rlnm[.]	Rotate Left Then AND With Mask	M	23	
rrib[.]	Rotate Right And Insert Bit	X	31	537
sf[o][.]	Subtract From	XO	31	8
sfe[o][.]	Subtract From Extended	XO	31	36
sfi	Subtract From Immediate	D	08	
sfme[o][.]	Subtract From Minus One Extended	XO	31	232
sfze[o][.]	Subtract From Zero Extended	XO	31	200
sl[.]	Shift Left	X	31	24
sle[.]	Shift Left Extended	X	31	153
sleq[.]	Shift Left Extended With MQ	X	31	217
sliq[.]	Shift Left Immediate With MQ	X	31	184
slliq[.]	Shift Left Long Immediate With MQ	X	31	248
sllq[.]	Shift Left Long With MQ	X	31	216
slq[.]	Shift Left With MQ	X	31	152
sr[.]	Shift Right	X	31	536
sra[.]	Shift Right Algebraic	X	31	792
srai[.]	Shift Right Algebraic Immediate	X	31	824
sraiq[.]	Shift Right Algebraic Immediate With MQ	X	31	952
sraq[.]	Shift Right Algebraic With MQ	X	31	920
sre[.]	Shift Right Extended	X	31	665
srea[.]	Shift Right Extended Algebraic	X	31	921
sreq[.]	Shift Right Extended With MQ	X	31	729
sriq[.]	Shift Right Immediate With MQ	X	31	696
srlmq[.]	Shift Right Long Immediate With MQ	X	31	760
srlq[.]	Shift Right Long With MQ	X	31	728



Mnemonic	Instruction	Format	Primary Opcode	Extended Opcode
srq[.]	Shift Right With MQ	X	31	664
st	Store	D	36	
stb	Store Byte	D	38	
stbrx	Store Byte Reverse Indexed	X	31	662
stbu	Store Byte With Update	D	39	
stbux	Store Byte With Update Indexed	X	31	247
stbx	Store Byte Indexed	X	31	215
stfd	Store Floating-Point Double	D	54	
stfdu	Store Floating-Point Double With Update	D	55	
stfdx	Store Floating-Point Double With Update Indexed	X	31	759
stfdx	Store Floating-Point Double Indexed	X	31	727
stfs	Store Floating-Point Single	D	52	
stfsu	Store Floating-Point Single With Update	D	53	
stfsux	Store Floating-Point Single With Update Indexed	X	31	695
stfsx	Store Floating-Point Single Indexed	X	31	663
sth	Store Half	D	44	
sthbrx	Store Half Byte Reverse Indexed	X	31	918
sthu	Store Half With Update	D	45	
sthux	Store Half With Update Indexed	X	31	439
sthx	Store Half Indexed	X	31	407
stm	Store Multiple	D	47	
stsi	Store String Immediate	X	31	725
stsx	Store String Indexed	X	31	661
stu	Store With Update	D	37	
stux	Store With Update Indexed	X	31	183
stx	Store Indexed	X	31	151
svc[l][a]	Supervisor Call	SC	17	
t	Trap	X	31	4
ti	Trap Immediate	D	03	
xor[.]	XOR	X	31	316
xoril	XOR Immediate Lower	D	26	
xoriu	XOR Immediate Upper	D	27	



---

# Chapter 3. Memory

## Chapter Contents

Virtual Memory .....	3-3
System Memory .....	3-3
Introduction .....	3-3
Description .....	3-3
Special Features .....	3-4
Memory Banks .....	3-5
DRAM Controller .....	3-5
Data Mux and Buffer .....	3-5
Bit Scattering .....	3-5
Memory Refresh .....	3-5







---

## 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. Subsequently, 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 system unit verifies whether that object is currently in system memory and, if so, supplies the appropriate (real) memory address. If 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 of 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 that memory closely associated with the system unit complex. The RISC System/6000 architecture provides for up to 4 gigabytes of system memory.

Direct Memory Access (DMA) operations to this memory neither synchronize nor update the system unit cache. This may cause the cache and its associated system memory to be inconsistent, resulting in the loss or corruption of data when the system unit and the I/O device both attempt to access the same memory area. The following set of guidelines should be followed to eliminate this problem:

- Ensure all cache (line) data has been flushed to system memory prior to starting an output DMA operation.
- If accessing a shared data area in system memory, addressing should be such that these accesses go through the IOCC buffer cache.
- On bus master input operations, unmap any shared memory pages by way of their controlling Translate Control Word (TCW) before attempting to use the data in system memory. A buffer flush operation can be performed at the same time a memory page is unmapped.

## Introduction

The RISC System/6000 memory board can be placed in either a SGR 2564 processor chip set or a SGR 2032 processor chip set. The board functions the same in both chip sets. In a SGR 2032 processor chip set, a single board is accessed at one time. Two 40-bit error checking and correction (ECC) words (32 data, 7 ECC and 1 redundant bit steering) can be transferred into or out of the board on a given clock cycle. ECC is not discussed in detail in this document since ECC is done in the cache and the Memory Control Unit (MCU). The memory board does not distinguish between ECC bits and data bits. As far as the memory board is concerned, a word is 40 bits wide.

Memory boards in the SGR 2564 processor chip set are operated in pairs. Two boards are accessed at one time, thus giving the system a 160-bit memory bus. The system memory to data cache transfers four 4-byte words on each memory access.

## Description

This section contains a general description of the RISC System/6000 memory board. The board is designed for 80 bit doubleword memory access. The size of the RISC System/6000 memory board ranges from 8 to 32 megabytes. Each board has three Application Specific



Integrated Circuit (ASIC) chips to control the Dynamic Random Access Memory (DRAM) and the data flow. There is one DRAM control chip and two Data Multiplexer chips. All address and timing signals from the MCU are input to the DRAM controller chip. Each Data Multiplexer (Mux) chip has two 40-bit bi-directional data buses connected to the memory banks. See Figure 21 for a functional block diagram of the 4-way interleaved memory board.

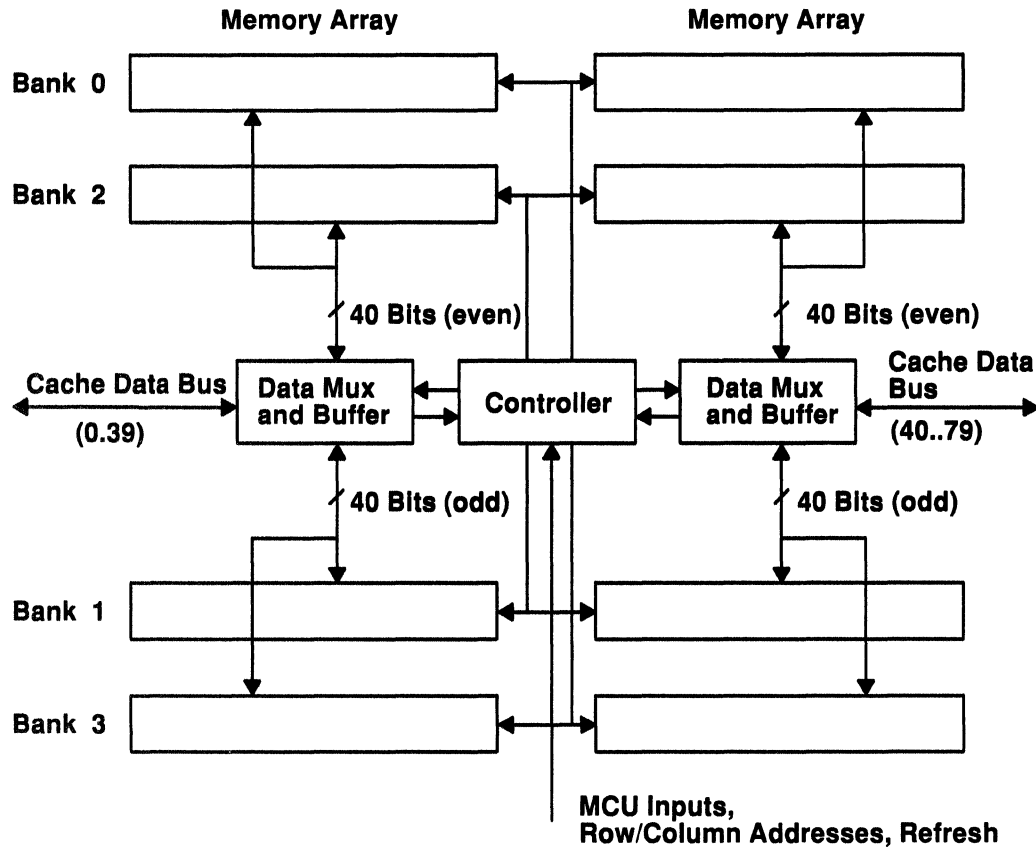


Figure 21. Memory Board Description

## Special Features

- High performance 80-bit data width
- 4-way interleaving
- High density (8, 16, and 32 megabytes)
- 5V dc and 3.6V dc DRAM power (Do not mix 5V dc and 3.6V dc memory boards in the same system unit.)
- Accepts generic timing inputs, generates multiple DRAM timing modes
- On-board refresh address counter
- Buffered read and write instructions
- Buffered write data
- Page mode operation



## Memory Banks

Each of the interleaved memory banks consist of two 40-bit wide memory arrays or Single In-line Memory Modules (SIMMs). The SIMMs are either 1M byte, 2M byte or 4M byte in density. The 1M byte SIMM has (10) 256K x 4 DRAMs surface mounted on one side. The 2M byte SIMM has (10) 256K x 4 DRAMs surface mounted on each side. The 4M byte SIMM has (5) 1024K x 4 DRAMs surface mounted on each side. The maximum board configuration has eight 4M byte SIMMs giving the board a total of 32M bytes. The minimum configuration has eight 1M byte SIMMs giving the board a total of 8M byte.

## DRAM Controller

The DRAM controller chip is responsible for providing all inputs to the DRAMs except data. This chip generates the necessary row-column addresses, row-column address strobes, read-write, and memory refresh timings. In addition, control logic for the Data Mux chips is generated in the DRAM controller. The inputs to the DRAM controller come from the Memory Control Unit (MCU) on the processor board, presence detect pins, and the Data Mux chips.

## Data Mux and Buffer

The Data Mux chips transfer data between the caches and the memory banks. Each Data Mux chip has a 40-bit data bus connected to the caches. Together, the two Data Mux chips provide the board with its 80-bit (double word) interface. The Data Mux chips have two 40-bit data buses, each of which is dotted to two memory arrays on a board in 4-way interleave mode. Figure 21 on page 3-4 shows a diagram of the memory board. An even data bus is connected to one SIMM in each of the even memory banks (banks 0 and 2), and an odd data bus is connected to one array in each of the odd memory banks (banks 1 and 3). During write operations, the Data Mux chips can buffer data as long as the DRAMs are being refreshed.

## Bit Scattering

The RISC System/6000 SGR 2564, SGR 3064 and SGR 2032 processor chip sets use bit scattering to ensure a minimum of bits of a memory word from being stored in a single DRAM. In the RISC System/6000 SGR2564 and SGR 3063 processor chip sets, there is no more than one bit from a single word stored in one DRAM. This is possible because the memory bus has four words on it and there are four bits in a given DRAM. In the RISC System/6000 SGR2032 processor chip set, there are two bits from each of the two memory bus words in a given DRAM. With a minimum of bits from a single word in a DRAM, the ECC is better able to detect and correct errors caused by a bad DRAM.

## Memory Refresh

The RISC System/6000 memory board is able to generate refresh timings. Every 15.6  $\mu$ s the board receives a refresh pulse from an external clock. When there are multiple memory boards, the refresh pulses from the clock are input to pairs of boards at 3.9  $\mu$ s intervals. Therefore, each board receives a refresh pulse every 15.6  $\mu$ s.

The DRAMs worst case requirement for a refresh signal is 4 ms for 1M byte and 2M byte SIMMs. In other words, each row address must receive a refresh signal every 4 ms. The 1M byte and 2M byte SIMMs have 512 row addresses. Since the refresh pulse comes every 15.6  $\mu$ s, it is required that with each refresh pulse two row addresses are refreshed. The necessary logic is provided in the controller to refresh two row addresses in the board when using 1M byte DRAMs. The 4M byte SIMM has 1024 row addresses but must be refreshed within 8 ms, so two refreshes must occur per refresh pulse. This too is implemented by the controller. It should be noted that with the double-side surface-mounted SIMMs, the corresponding row addresses for DRAMs on both sides of the board are refreshed together.







---

# Chapter 4. System I/O Structure

## Chapter Contents

Description .....	4-3
System Structure .....	4-4
Virtual Memory .....	4-5
System Memory .....	4-6
Bus Memory .....	4-6
Bus I/O .....	4-6
IOCC Control Registers .....	4-6
Data Security .....	4-8
Bit and Byte Numbering Conventions .....	4-9
Processor and Bus Notation .....	4-9
Byte Reversal .....	4-13
I/O Bus Protocols .....	4-15
Arbitration .....	4-15
Priority Assignment .....	4-17
Non-Preemptive Burst .....	4-17
Preemptive Burst .....	4-17
Fairness Modes .....	4-18
Basic Transfer Cycle .....	4-18
Streaming Data .....	4-19
Dynamic Bus Sizing .....	4-19
Partial Transfer Cycles .....	4-21
Bus Refresh .....	4-21
Bus Errors .....	4-21
Invalid Address .....	4-21
Parity Errors .....	4-22
Channel Check .....	4-22
Bus Time Out .....	4-22
Interrupt .....	4-22
Programming Model .....	4-23
Load and Store Instructions .....	4-23
Address Spaces and Effective Addresses .....	4-24
I/O Segment Register Definition .....	4-27
Address and Data Alignment .....	4-29
String Operations .....	4-30
Load and Store Access Authority Checking .....	4-30
Load and Store Error Conditions .....	4-32
Translation, Protection, and the TCW Table .....	4-34
Bus Master .....	4-37
Buffered Bus Master .....	4-37
Unbuffered Bus Master .....	4-42
Bus Master Access Authority Checking .....	4-44
Bus Master Error Conditions .....	4-45
DMA Slave .....	4-46
DMA Slave Operations Using Tags .....	4-47



DMA Slave Operations Using TCW's .....	4-52
DMA Slave Bus Protocols .....	4-56
DMA Slave Transfers to Bus Memory .....	4-56
DMA Slave Transfers to System Memory .....	4-56
Special Sequences .....	4-57
DMA Slave Error Conditions .....	4-57
IOCC Commands .....	4-59
Time Delay Command .....	4-59
End of Interrupt .....	4-61
Lock Command .....	4-61
Enable and Disable Commands .....	4-62
Buffer Flush Commands .....	4-63
Bus Master Buffer Flush Command .....	4-63
DMA Slave Buffer Flush Command .....	4-64
Buffer Invalidate Command .....	4-65
I/O Interrupts .....	4-65
Special Facilities .....	4-70
Board Configuration Data .....	4-71
IOCC Configuration Register .....	4-71
Bus Status Register .....	4-75
TCW/Tag Anchor Address Register .....	4-76
Component Reset Register .....	4-77
System I/O and Standard I/O .....	4-78
System I/O .....	4-78
System Registers .....	4-78
Nonvolatile RAM .....	4-78
Standard I/O .....	4-78
Exception Reporting and Handling .....	4-80
Implementation Details .....	4-80
IOCC Configuration Register .....	4-80
System Registers .....	4-81
Nonvolatile RAM .....	4-82
Standard I/O .....	4-84
Bus Master Transfers .....	4-84
Component Reset Register .....	4-84
Notes on Error Detection .....	4-84
Bus Timeout .....	4-84
I/O Interrupts .....	4-84
Lock Command .....	4-85
Power-On Reset .....	4-85
IPL Procedures .....	4-85
Architectural Deviations .....	4-88



---

## Description

This chapter describes the RISC System/6000 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 4-78.

The IOCC design allows certain variations of function and performance to optimize its usage across multiple machine environments. The specific personalization is established with the contents of the IOCC Configuration register (See "IOCC Configuration Register" on page 4-71 and "Implementation Details" on page 4-80.)

Figure 22 illustrates the logical view of the IOCC in the RISC System/6000 units.

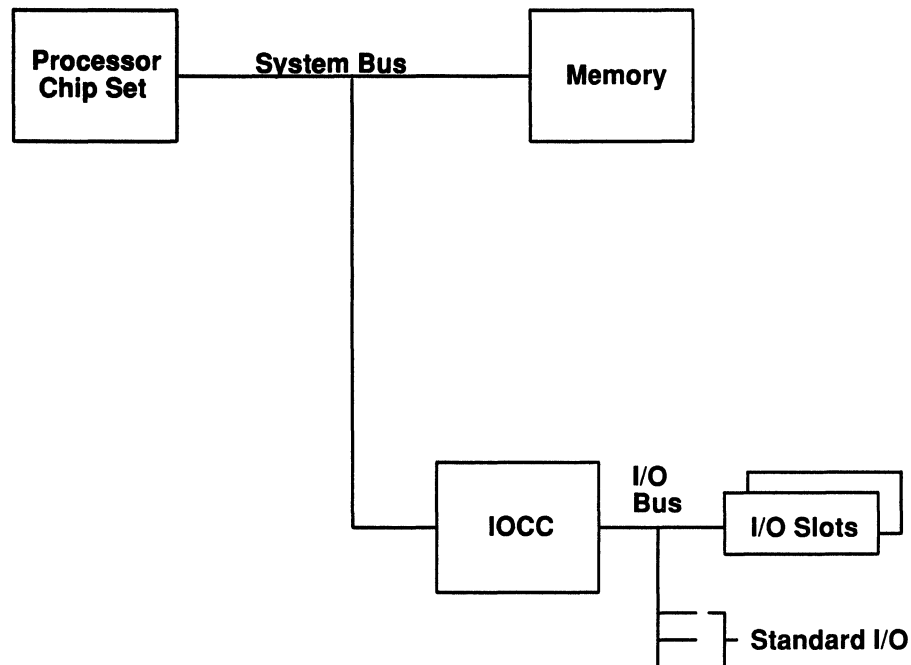
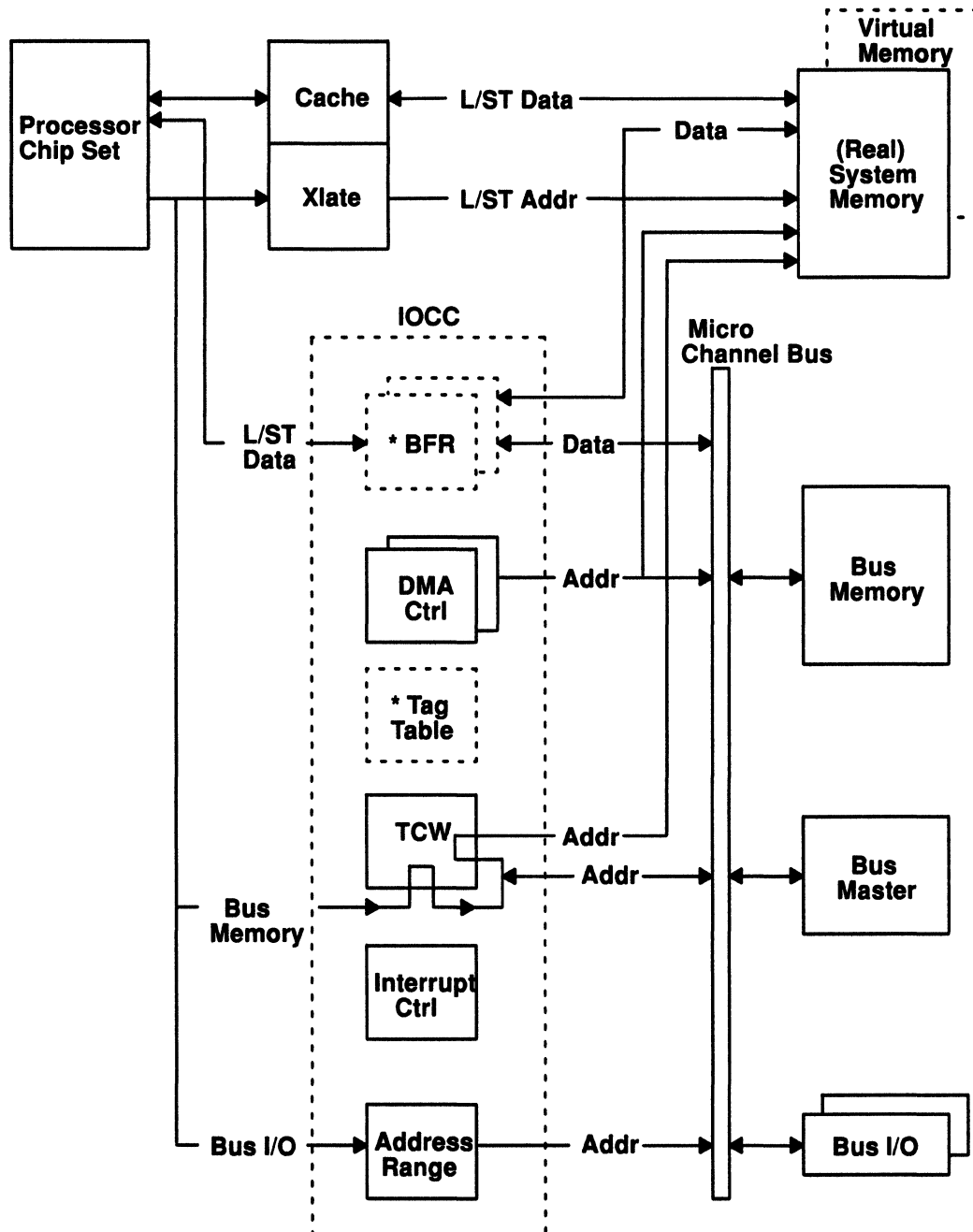


Figure 22. System Block Diagram



## System Structure

Figure 23 illustrates a more detailed logical view of the RISC System/6000 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 4-80).

Figure 23. 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. Mapping of the virtual address to system memory is then always managed 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 those devices for which they are authorized to use. Accesses to bus I/O are verified by way of an address range check, and 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 4-34.

The RISC System/6000 I/O architecture includes the definition of 16 independent I/O channels. One channel (X'F') is reserved for use 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 4-71 and "Implementation Details" on page 4-80.) 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 RISC System/6000 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 of this mode (called the buffered mode) and the amount of IOCC buffer is implementation specific. (See "IOCC Configuration Register" on page 4-71 and "Implementation Details" on page 4-80.)

Normally, all accesses to system memory go through the processor chip set cache. However, if sharing memory areas with I/O devices, means must be provided for maintaining cache coherency. How cache coherency is provided is implementation specific. (See "IOCC Configuration Register" on page 4-71 and "Implementation Details" on page 4-80.) 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. Pages in the bus memory address space are mapped to system memory by way of the TCW table. Mapped pages are checked for proper access authority before allowing an access to proceed. Since the IOCC cannot intercept or stop accesses 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 RISC System/6000 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 4-71 and "Implementation Details" on page 4-80 .

## **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 and 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 and, if so, supplies the appropriate (real) memory address. If 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 that memory closely associated with the processor chip set complex. The RISC System/6000 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 chip set cache or Page Frame Table (PFT). This can cause the cache and its associated system memory to be inconsistent, resulting in the loss or corruption of data when the processor chip set and an I/O device both attempt to access the same memory area.

In buffered mode, it is the responsibility of the software to ensure there is no data lost due to cache inconsistencies. In unbuffered mode, the hardware maintains the cache coherency. Buffered versus unbuffered modes are described later in this chapter.

## **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 RISC System/6000 units, these two address spaces are mapped together as illustrated in Figure 33 on page 4-23. This address space is differentiated from the I/O address space by way of 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 system cache and do not suffer from cache consistency problems described in "System Memory" on page 4-6.

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. Refer to the "Translation, Protection, and TCW Table" section on page 4-34 for a description of this mapping process. These accesses 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 "System Memory" on page 4-6 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 includes 16 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 illustrated in Figure 24 on page 4-9.

## **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 and are addressed in a disjoint address space inaccessible to I/O bus devices. This address space is defined in



such a way that it may be mapped onto the I/O bus, providing implementation flexibility in distributing IOCC control facilities across multiple chip packages. Refer to the “Special Facilities” section on page 4-70 for a description of these registers.



## **Data Security**

The RISC System/6000 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, and 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

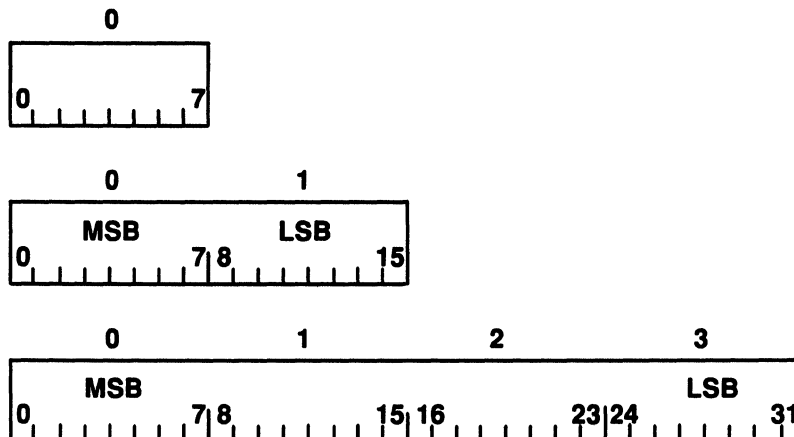
This section describes the processor and bus notation and the byte reversal numbering conventions.

### Processor and Bus Notation

Standard IBM notational practice is to address multi-byte fields in ascending order from left to right. This results in the most significant byte (MSB) always having the lowest address and provides consistency in addressing which 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 Micro Channel reverses both bit and byte addressing notations (The reason for this is historic and is based on the vendor processors which were used in Micro Channel machines when the Micro Channel Architecture was developed). Figure 24 illustrates the notational differences between the Micro Channel and RISC System/6000 family.

#### IBM Notation



#### Micro Channel Notation

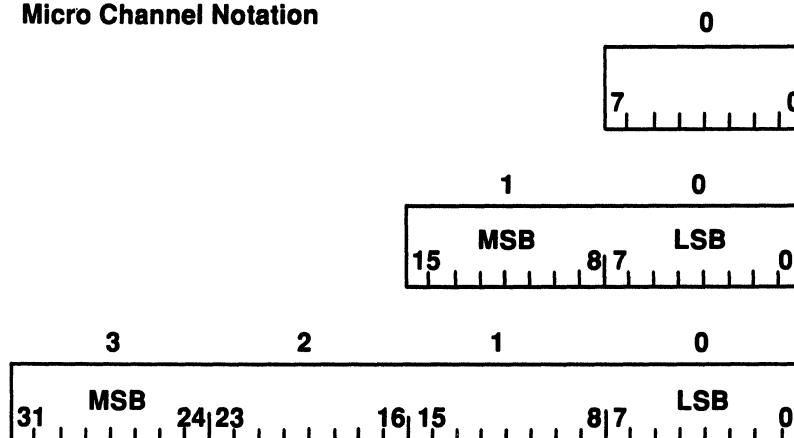


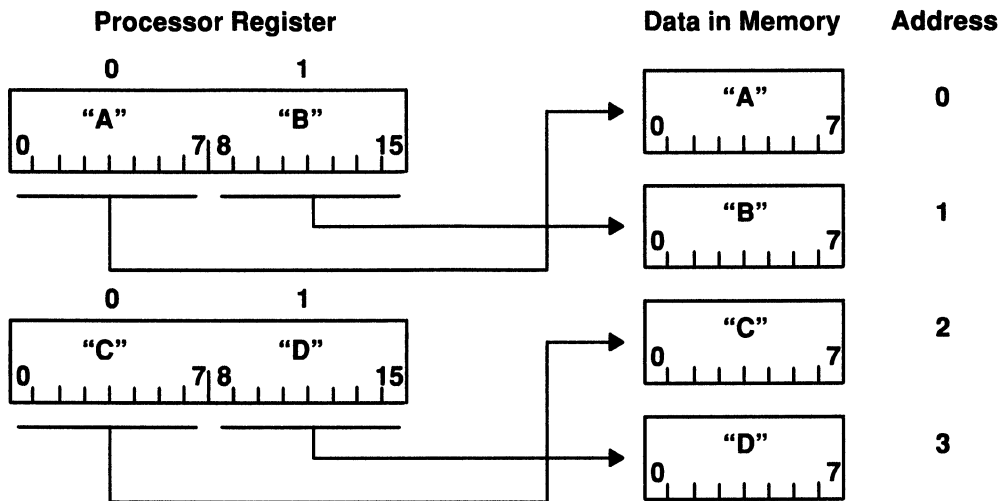
Figure 24. Data Addressing and Bit Numbering Notations



The IBM Micro Channel 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 25 on page 4-11 illustrates the consistency with the standard IBM notation and Figure 26 on page 4-12 illustrates the address inconsistency when using the Micro Channel notation. With the Micro Channel 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.

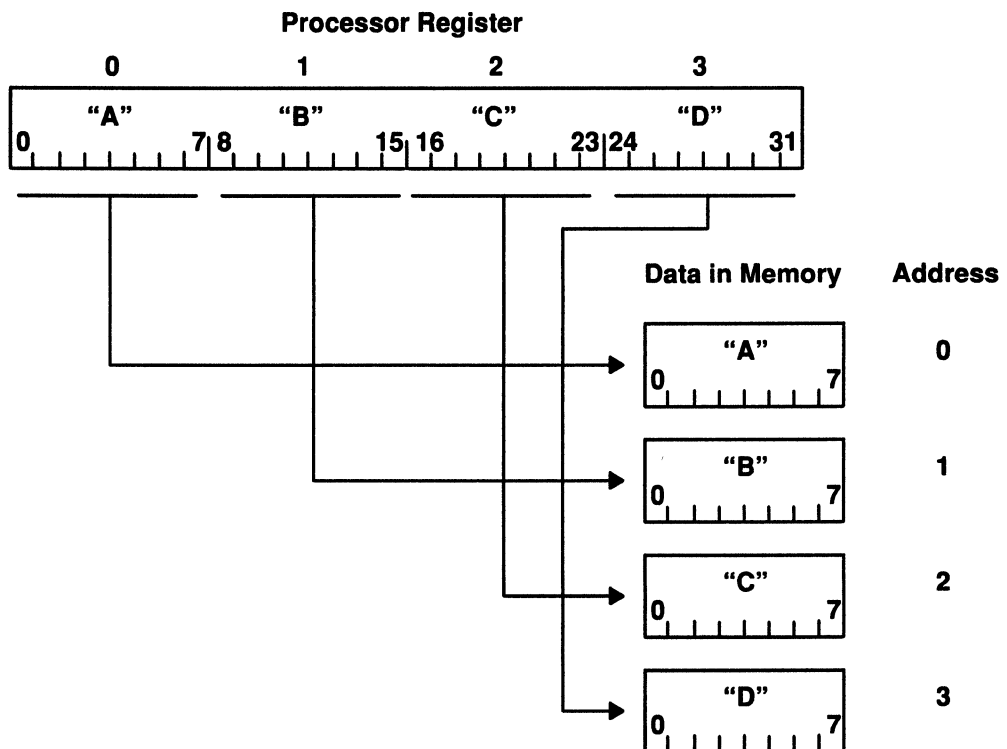
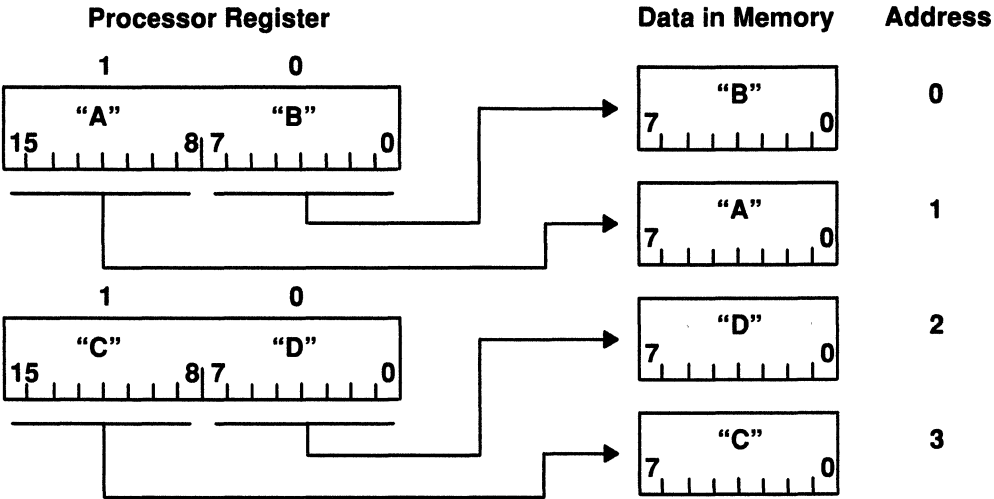


Figure 25. Addressing Consistency Using Standard IBM Notation



Two half-word Store instructions from the processor register to memory.



Full-word Store instruction from the processor register to memory.

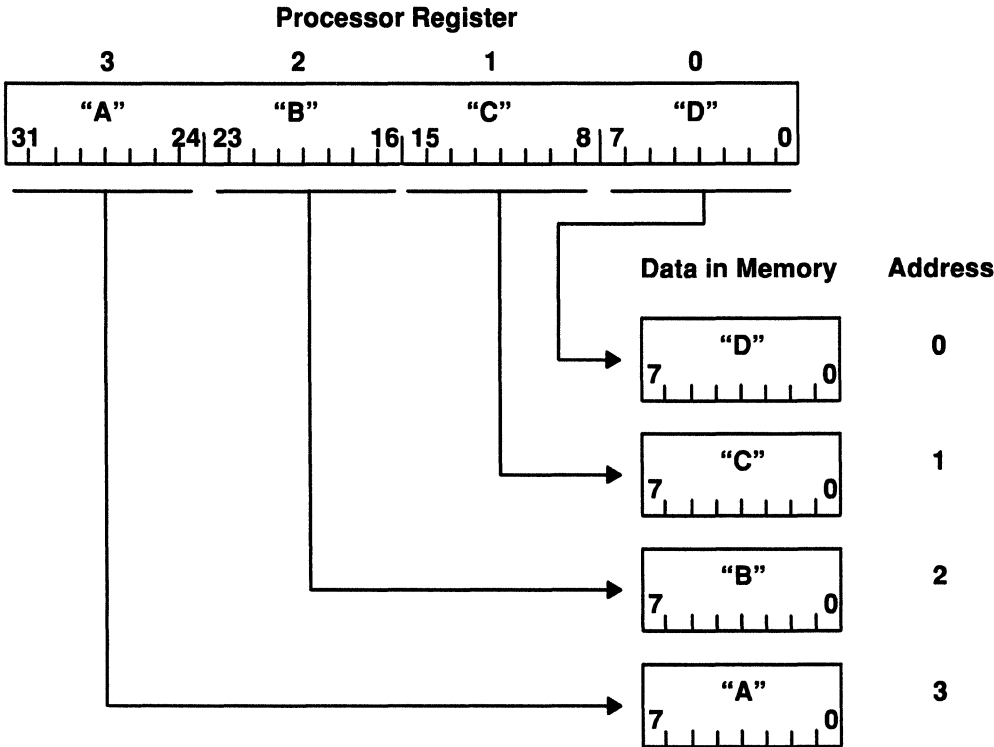


Figure 26. Addressing Inconsistency When Using Micro Channel Notation



## Byte Reversal

Data in the RISC System/6000 unit is handled by the compilers using standard IBM addressing notations. To meet addressing notations of the IBM Micro Channel, the byte ordering must be reversed. The IOCC and the system board are designed to provide byte-order reversal as illustrated in Figure 27. This reversal occurs in both directions as information passes through the IOCC.

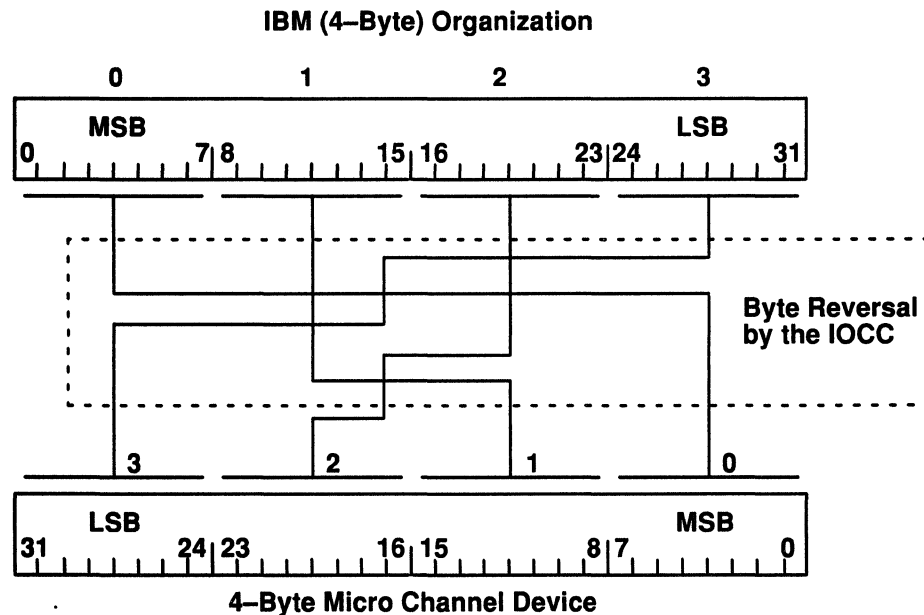
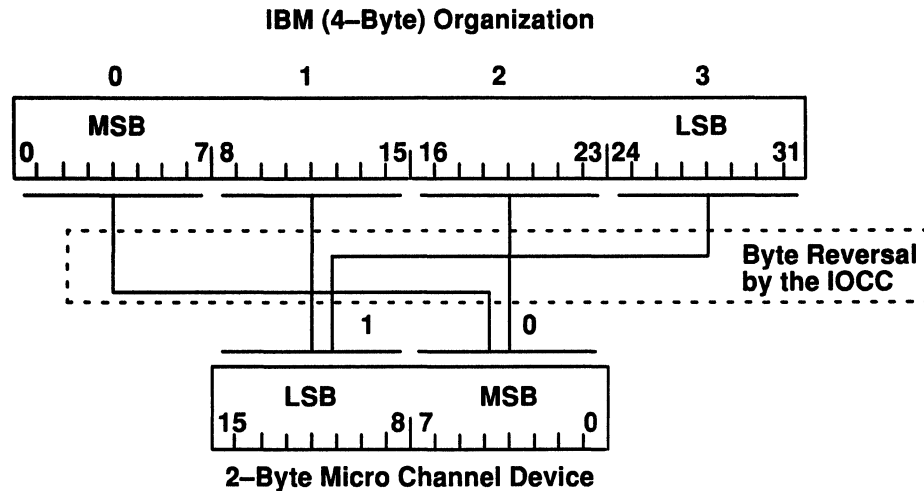


Figure 27. PC Bus Byte Reversal

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

Combining Figure 26 and Figure 27 gives the example shown in Figure 28.



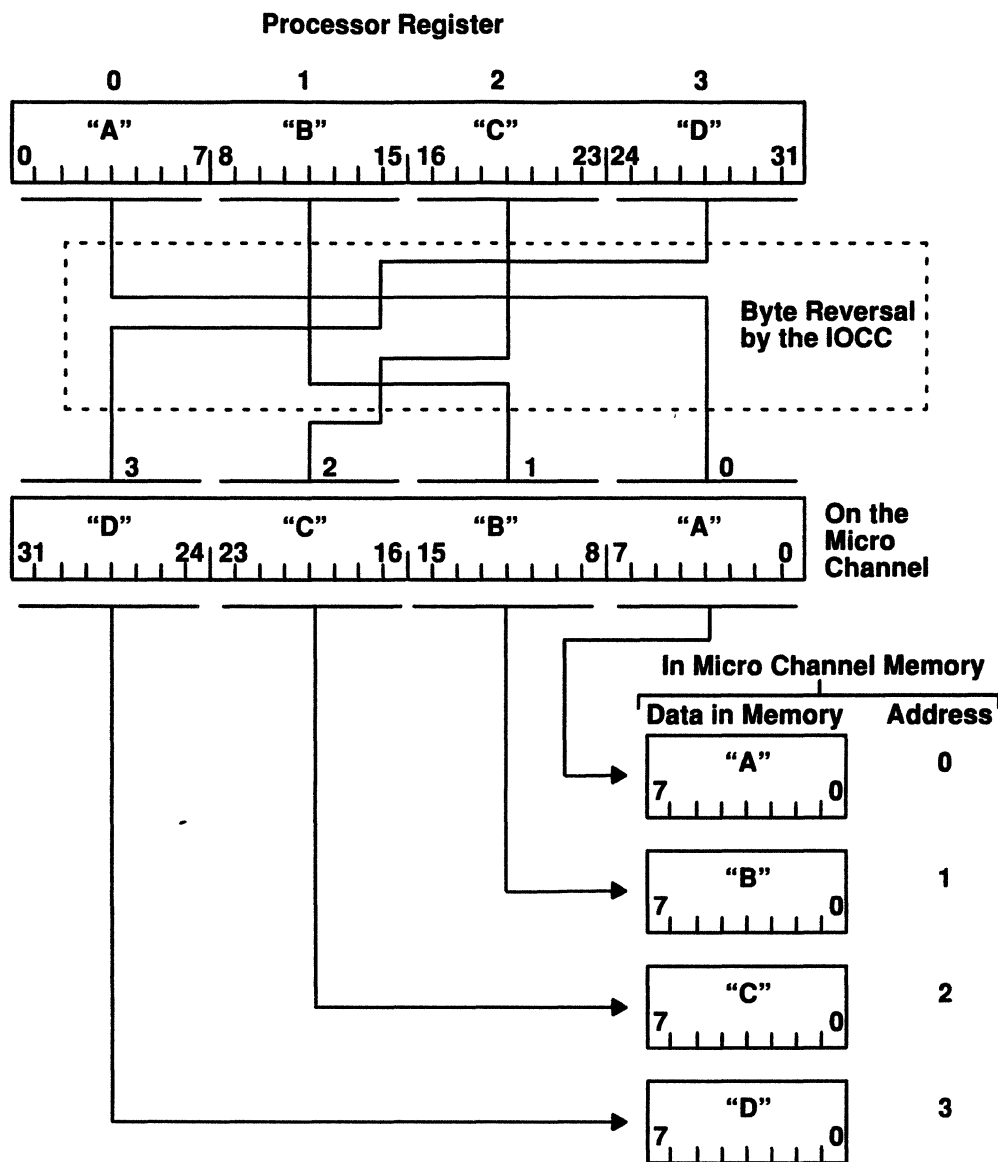


Figure 28. Example Showing Micro Channel Byte Reversal



---

## I/O Bus Protocols

The RISC System/6000 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 more details, see the *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture*.

**Note:** This chapter uses the abbreviated signal names as they appear in the *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture*; 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
- Multi-drop (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
- Extendable to multiple buses.

The arbitration mechanism distributes prioritization among the arbiters but retains control and clocking functions within the IOCC. It uses a bus-like structure and does not require any slot-unique wiring for its operation. Bus arbitration timing is programmable and is established by a field in the IOCC Configuration register.

Figure 29 illustrates 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.



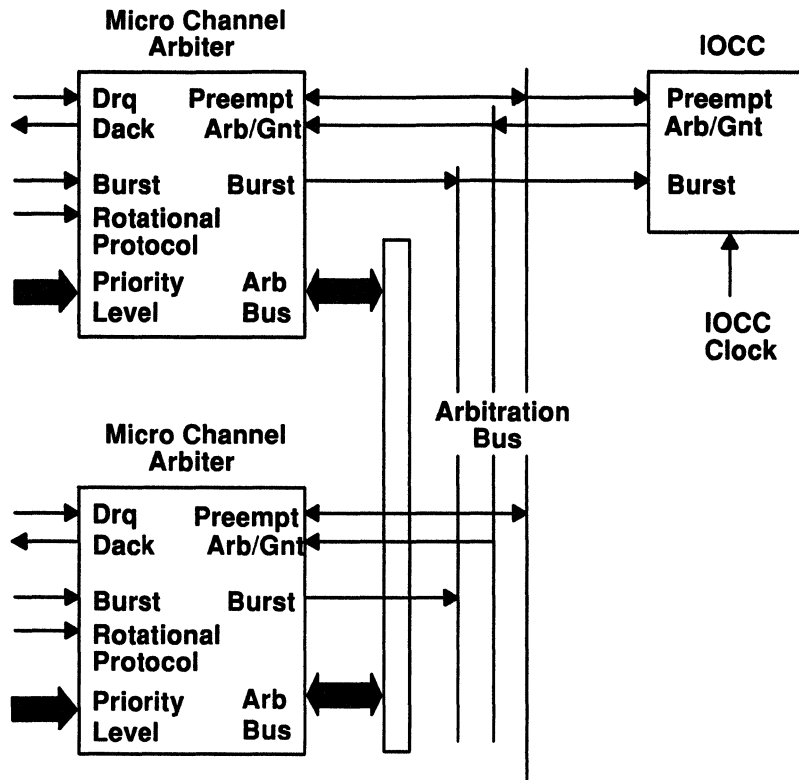


Figure 29. I/O Bus Arbitration

Figure 30 illustrates 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. This bus is designed in such a way that lower priority devices remove themselves from contention, and only the highest priority requester is left on the bus after a logic settling time. The IOCC then reactivates the 'arb/gnt' signal, and the device with its arbitration level value on the arbitration bus is granted use of the bus. Device Request (Drq) and Device Acknowledge (Dack) are signals (internal to each of the device arbiters) which signal a request to arbitrate for the bus, and acknowledgement of being granted the bus, respectively.

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'.



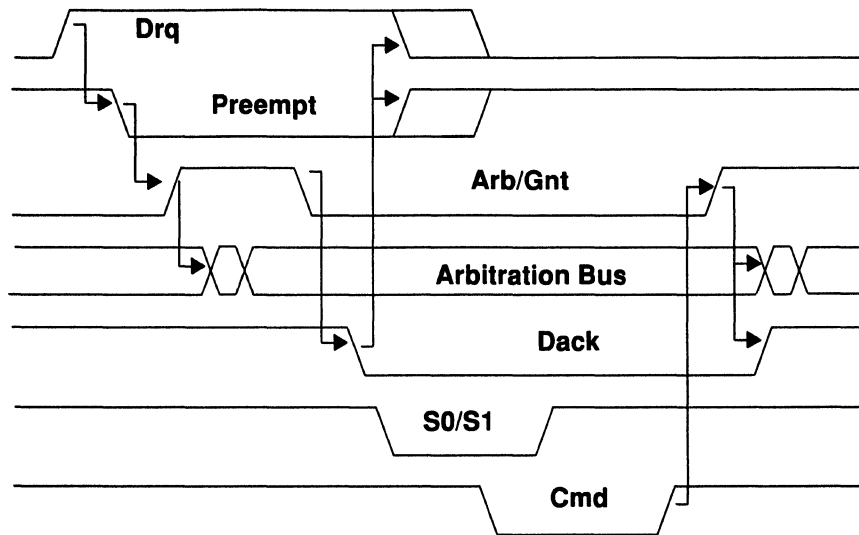


Figure 30. Arbitration Cycle

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 have to arbitrate for the bus for I/O Load and Store instructions.

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 deactivated, 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.

### Non-Preemptive Burst

Devices can force non-preemptive 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 non-preemptive for up to 7.8



microseconds following a preemption request. This 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 is provided to make 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 non-preemptive for up to 7.8 microseconds following a preemption request.

## Basic Transfer Cycle

Although the RISC System/6000 I/O architecture is generic and can attach a number of unique buses, the intended design point is the Micro Channel. These bus protocols are illustrated in Figure 31.

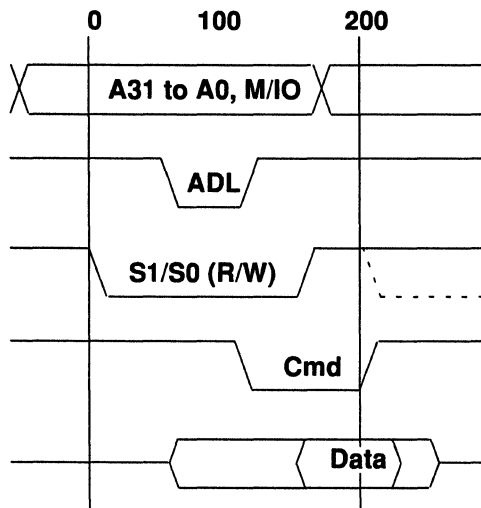


Figure 31. I/O Bus Cycles

The Micro Channel offers a 32-bit data path with 4G bytes of address space. It includes extensive support for reliability, availability, serviceability, extendability, 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 IOCC initiated transactions such as 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 transfer data, with data being clocked onto the bus on one clock edge and clocked off the bus on the next clock edge. The operation proceeds with new data being 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 *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture*.

## 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 discrepancies in data transfer widths be automatically managed by the current bus master. 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 multicycle operations in the RT system and as dynamic bus sizing in the Micro Channel architecture. The multiple I/O cycles complete as a preemptable operation in the RISC System/6000 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 illustrated in Figure 32.

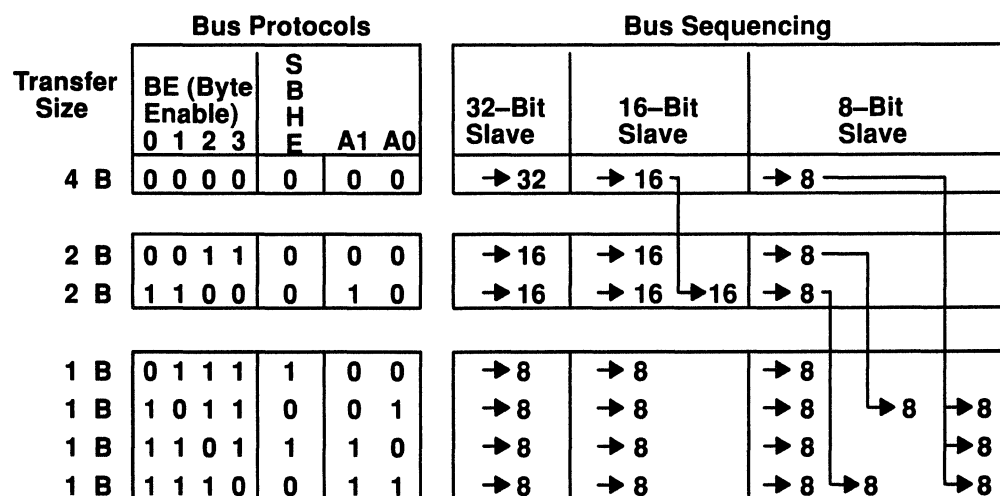


Figure 32. Dynamic Bus Sizing



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 the "String Operations" section on page 4-30 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 and 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 non-aligned addresses transfer the data to the device using multiple (partial write) cycles. These write operation 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 non-aligned, 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 4-30 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 4-71 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 24 on page 4-9) are incremented by one, and a copy is placed on the Micro Channel 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 4-32 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 sdbk' 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 4-27 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 4-80 for details of the RISC System/6000 I/O parity support.

## Channel Check

The Micro Channel includes a 'chck' signal which 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 RISC System/6000 unit, see "Exception Reporting and Handling" on page 4-80.

It is important to note that the RISC System/6000 unit is designed to recover from synchronous channel checks. Adapters that use the 'chck' signal asynchronously will make an Initial Program Load (IPL), the only recovery which 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. For implementation details, see "Implementation Details" on page 4-80.

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. 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. 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 IOCC provides pull-up resistors on all Micro Channel interrupt signals so that unused lines float to the inactive state. Refer to the "I/O Interrupt" section on page 4-65 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 to those 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 33 illustrates this 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.

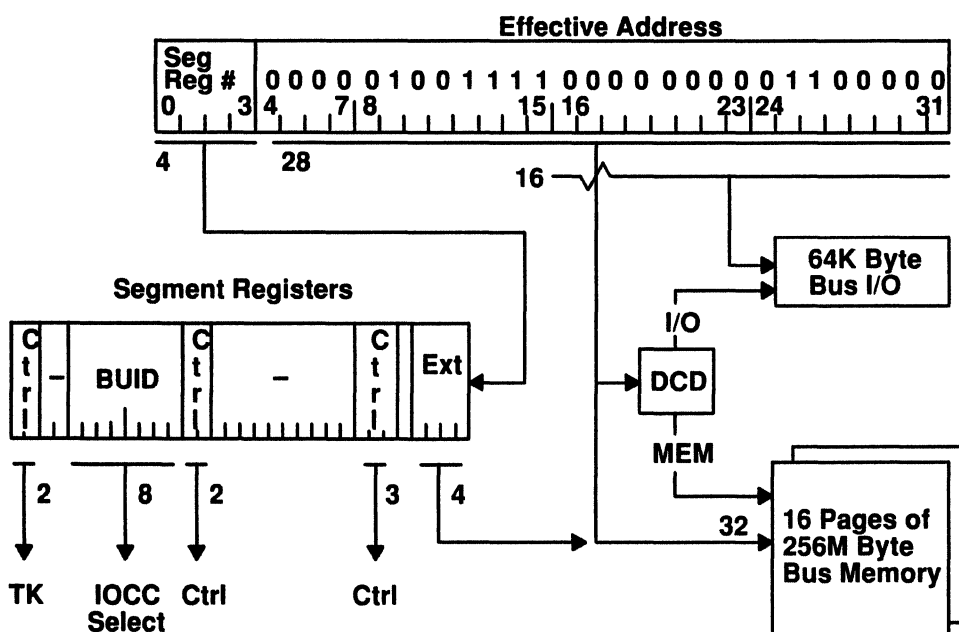


Figure 33. I/O Addressing



## Address Spaces and Effective Addresses

Figure 34 illustrates the RISC System/6000 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 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 (4 G bytes of total address space), and separate Segment registers must be used to address across pages. If consecutive Segment registers are used when crossing bus memory pages, the addressing is continuous, and appears as a single linear address space. 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 35 on page 4-26.

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 RISC System/6000 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 64 KB 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 36 on page 4-26.

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,  $T = 1$ ,  $I = 1$ ,  $M = x$ , and  $EXT = x$ .

Included in this address space are IOCC registers, the tag and TCW tables, the system registers and Nonvolatile Random Access Memory (NVRAM). Note that some references to the IOCC control space are on word boundaries only and require a data length of 4 bytes, for example the tag and 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 an 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 37 on page 4-27.

Although bus memory and bus I/O are disjointed in PC products, the RISC System/6000 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 illustrated in Figure 34. The values for the T, I and M bits for each of the three I/O effective address operating modes are described previously and are illustrated in Figure 34.

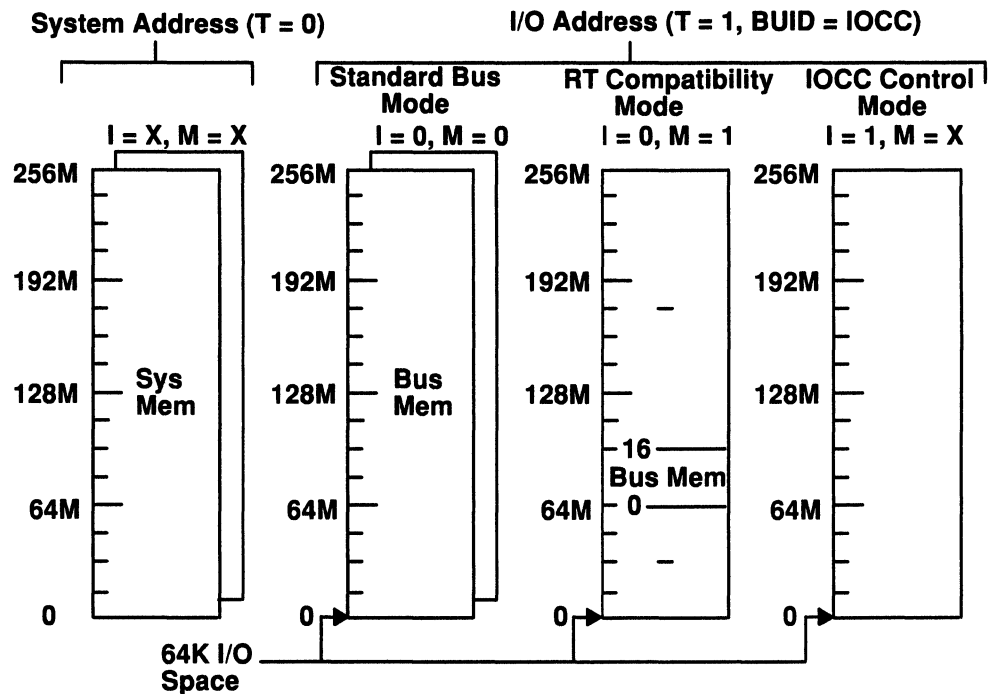


Figure 34. Addressing Model



Figures 35, 36, and 37 summarize the RISC System/6000 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.

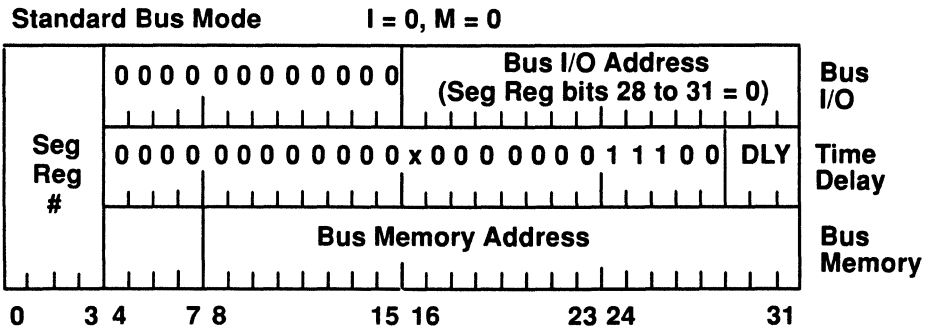


Figure 35. User Effective Addresses: Standard Bus Mode

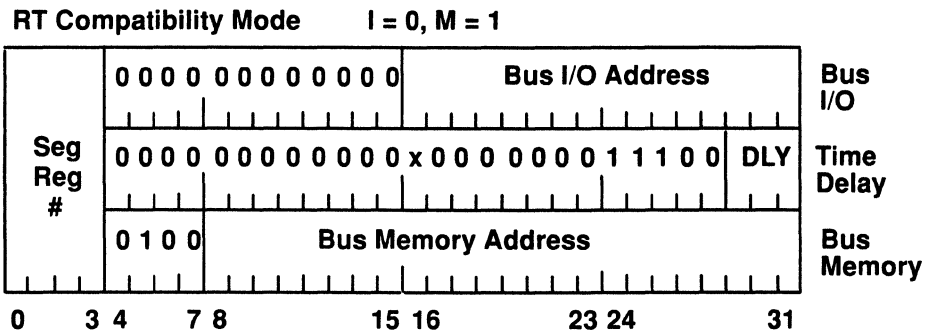
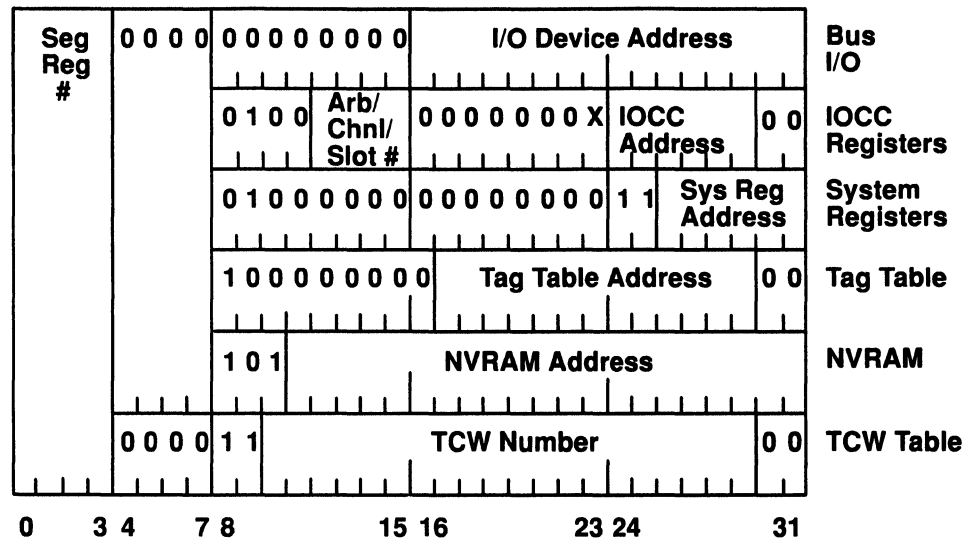


Figure 36. User Effective Addresses: RT Compatibility Mode



# IOCC Addressing

I = 1, M = X



# IOCC Commands

I = 1, M = X

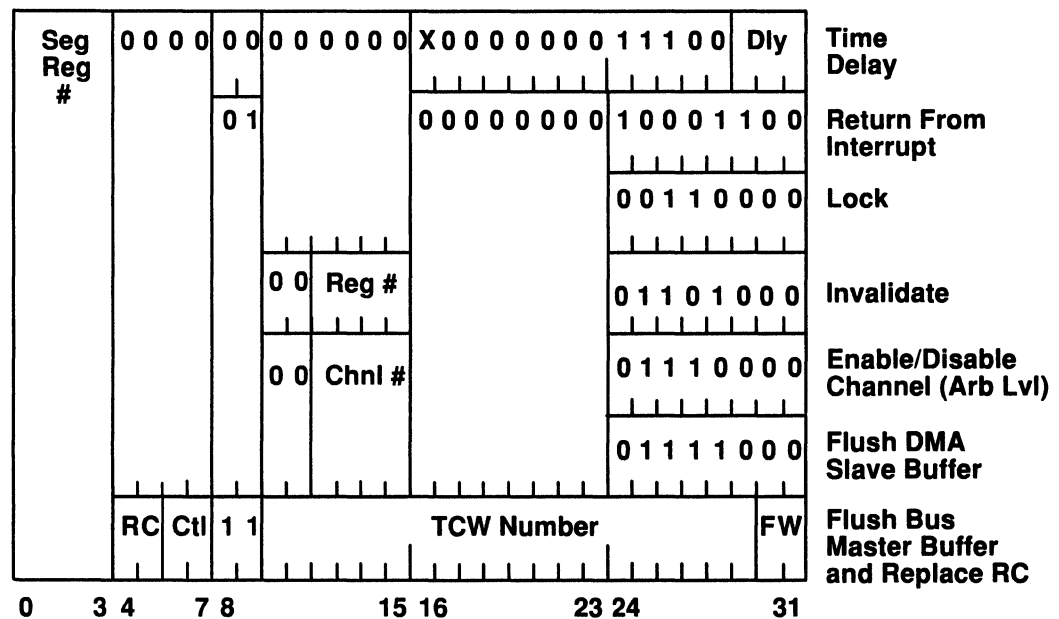


Figure 37. IOCC Effective Addresses

## 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 38.



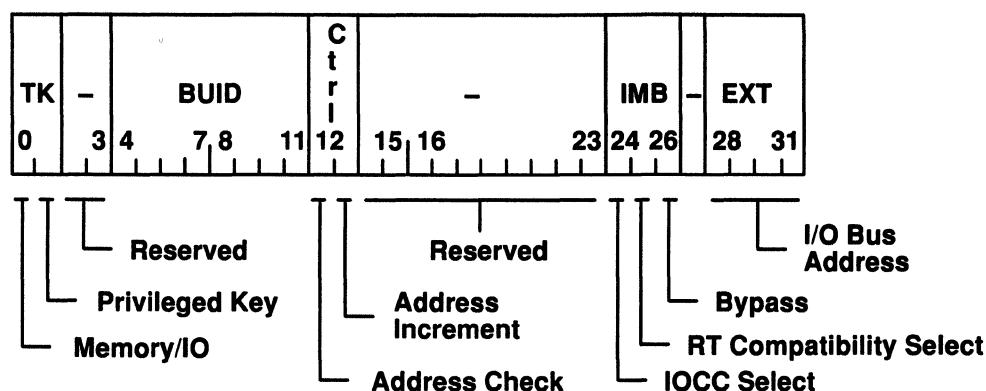


Figure 38. I/O Segment Register

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 38, 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 the "Invalid Address" section on page 4-21. 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 39 summarizes all the combinations of bit 12 and the address response by an I/O board.



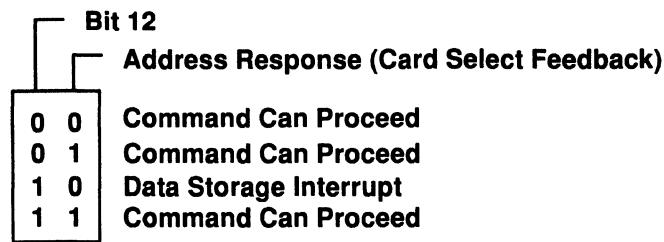


Figure 39. Bit 12 and Address Response Definition

- 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 and this bit controls whether the address is incremented between the I/O bus cycles. See the “Dynamic Bus Sizing” section on page 4-19 for a description of this function. Addresses are incremented when bit 13 is set to a value of 1 and 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. Caution should be used in 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 and 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 I equals 1.
- 27** Reserved: This bit is reserved and must be set to a value of 0.
- 28–31** Bus Memory 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 4-21 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 will be 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 4-71 and "Implementation Details" on page 4-80).

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 4-19 and "Partial Transfer Cycles" on page 4-21 for details of these functions. Slower than expected I/O instruction processing can have detrimental effects on system performance. For example, the system processor can not 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 illustrated in Figure 40. 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.



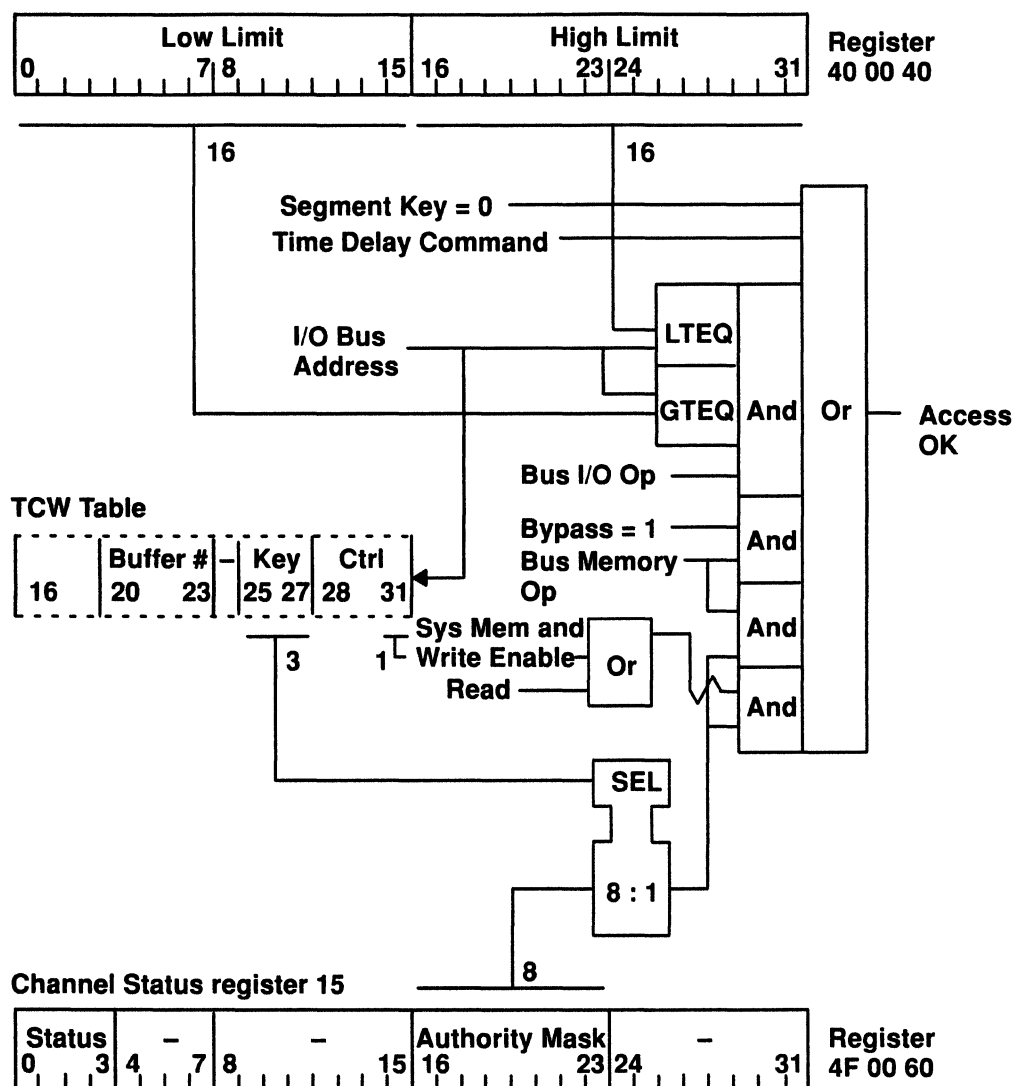


Figure 40. Load and Store Access Authority Checking

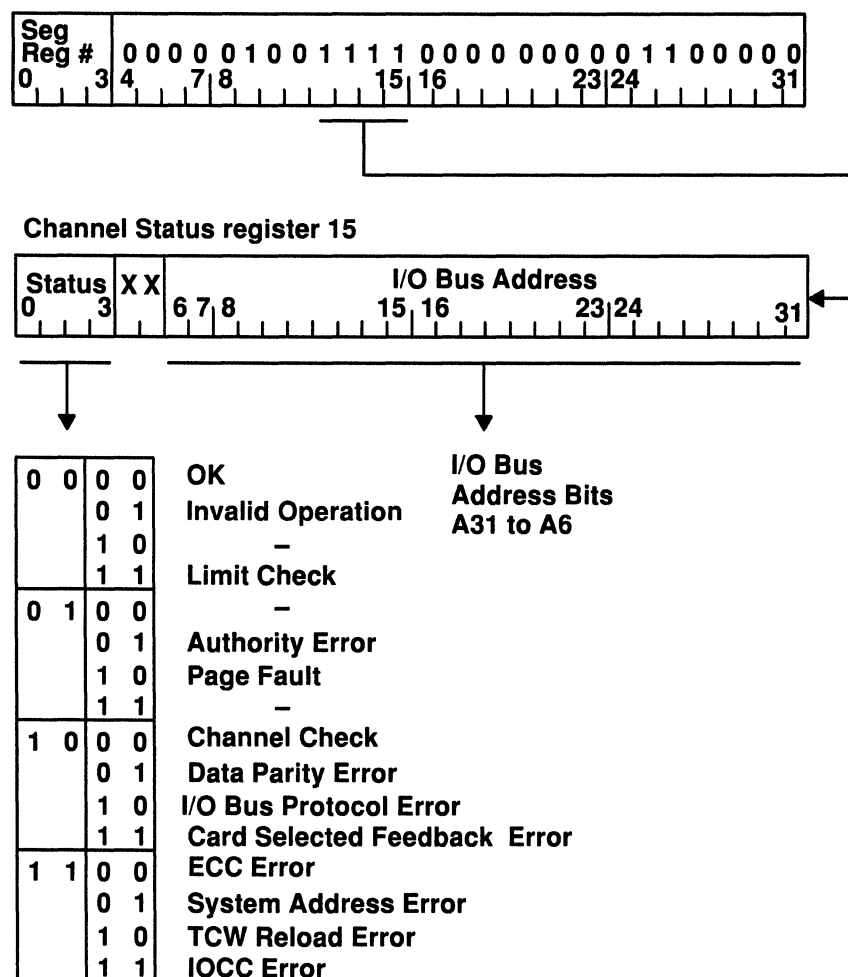
Operations to bus I/O have fine address granularities and are verified by way of address range checking. Address ranges are controlled by the operating system and restrict access of user programs to authorized devices. Address range information is considered part of the user (program) context and is 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 4-59 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



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.

Error conditions that arise in Load and Store instructions include bus errors, programming errors, and hardware errors. The specific cause of the error is determined by examining the error code contained in Channel Status register 15. On a memory error, I/O bus (page) address bits A31 to A6 are placed in bits 4 to 31 of this register. This assists in determining the cause of error. Figure 41 illustrates the resultant register contents.





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 the "I/O Interrupts" section on page 4-65. Refer to "Exception Reporting and Handling on page 4-80 for more information. Load and store error codes are summarized as follows:

<b>Error Code</b>	<b>Description</b>
<b>0 0 0 1</b>	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).
<b>0 0 1 1</b>	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.
<b>0 1 0 1</b>	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.
<b>0 1 1 0</b>	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.
<b>1 0 0 0</b>	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 (card selected feedback error takes precedence over channel check error).
<b>1 0 0 1</b>	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.
<b>1 0 1 0</b>	I/O Bus Protocol Error: This error code is set if a Micro Channel protocol error has been detected (for example, a card pulls the 'cd ds 32' line on the Micro Channel but does not pull the 'cd ds 16' line at the same time).
<b>1 0 1 1</b>	Card Selected Feedback Error: This error code is set if, after a device is addressed, it does not respond by driving the 'cd sdbk' line, and the address check bit is on in the I/O Segment register. Conditions which could cause this to occur are if the device is not present, if the device is not seated in the card slot properly, if the device is not enabled or if the device detects bad address parity and does not respond to that address. This error code takes precedence over a channel check.
<b>1 1 0 0</b>	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 is similar to a bus master operation).
<b>1 1 0 1</b>	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.



- |                |  |
|----------------|--|
| <b>1 1 1 0</b> | <b>TCW Reload Error:</b> 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).   |
| <b>1 1 1 1</b> | <b>IOCC Error:</b> 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 or tag table access or <b>flush</b> 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. Any subsequent load or store errors will result in Data Storage interrupts, but do not change any condition status in the Channel Status register.

Load and Store instructions 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.

Channel 15 is treated differently than the other channels following an error. Channel 15 always remains enabled, or a deadlock situation would exist. All other channels are disabled following an error.

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 4-80) and a bus timeout condition (see “Bus Timeout” on page 4-22 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 4-80 for more information.

## 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 4-30 for a detailed description.

The TCW table organization is illustrated in Figure 42. 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 and varies from a minimum of 96 K bytes (maps 96M bytes of bus memory) to a maximum of 4 M bytes (maps the full 4G bytes bus memory space). A field in the IOCC Configuration register is used to specify the amount of TCW RAM supplied. Refer to "IOCC Configuration Register" section on page 4-71 for details. Access to the TCW table entries must be 4-byte aligned.

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.

The TCW table is a protected system resource located in the IOCC address space between addresses X'0 C0 00 00' and X'0 FF FF FF'. 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.

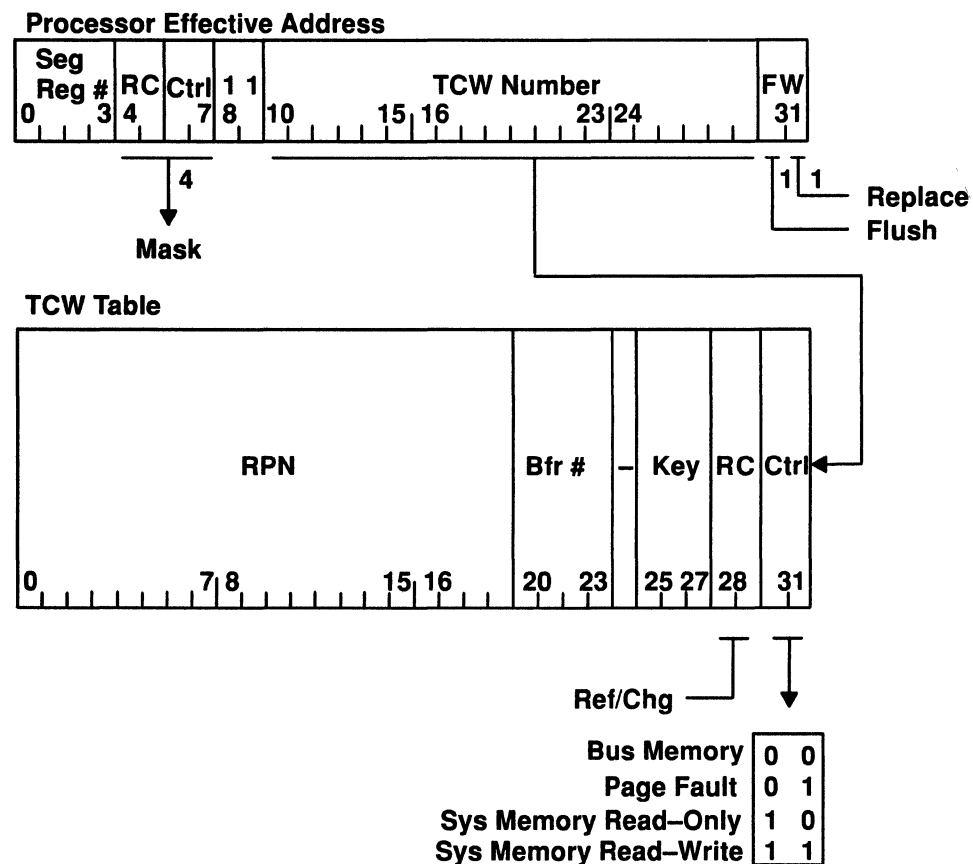


Figure 42. TCW Table

TCW's can be used for both bus master and DMA slave operations. A detailed description of a TCW entry is described 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.



- 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, caution should be exercised 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.
- 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 4-30 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 4-44 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 43.

30	31	
0	0	<b>Bus Memory</b>
0	1	<b>Page Fault (No Access)</b>
1	W	<b>System Memory</b>

Figure 43. 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 4-80.) 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 will result 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) bit in the Segment register overrides bit 31, that is, privileged access is not limited by the Read-Write or Read-Only bit.

## **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 RISC System/6000 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 4-71 and "Implementation Details" on page 4-80).

### **Buffered Bus Master**

Figure 44 illustrates 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.

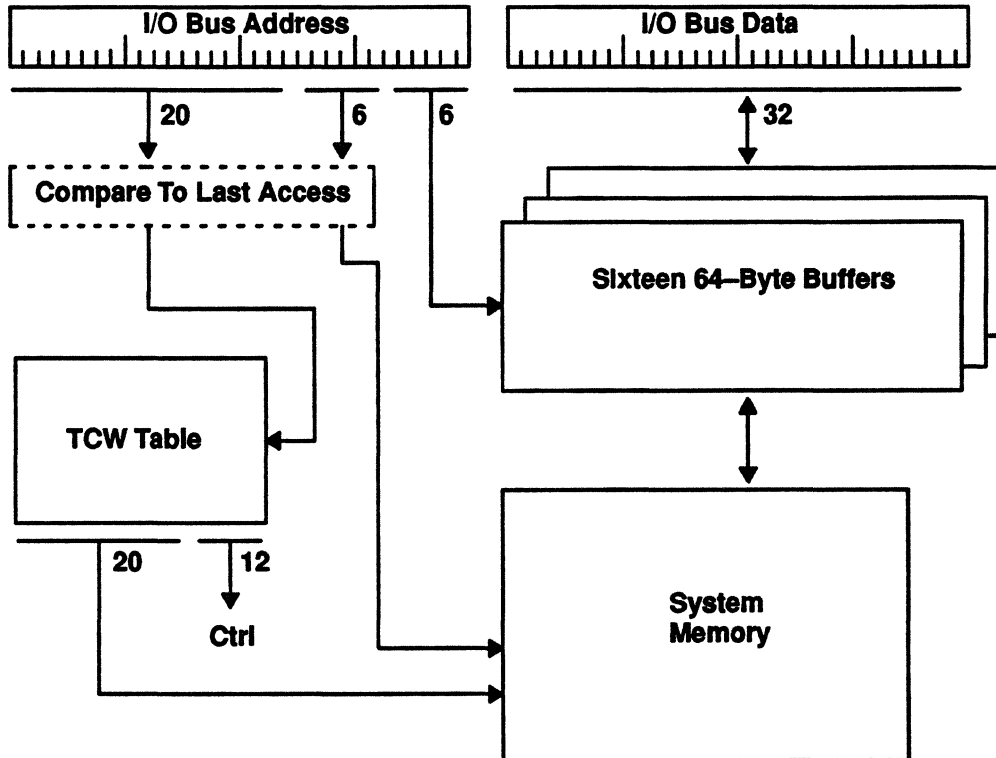


Figure 44. Buffered Bus Master Data Transfer Operation

As illustrated in Figure 45, 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.



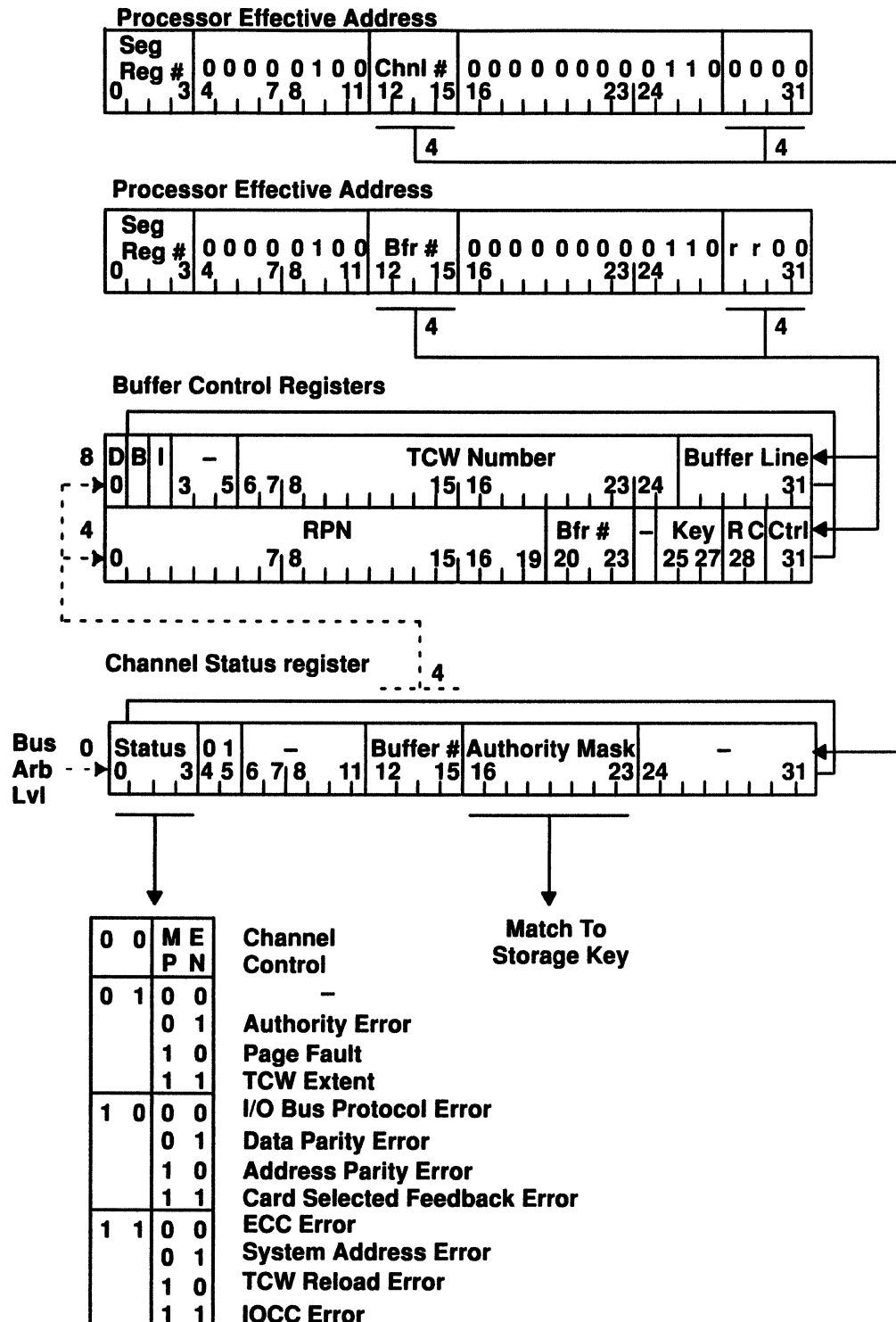


Figure 45. Buffered Bus Master Control Registers

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 4-45 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). Bit 3 is controlled by channel <b>enable</b> and <b>disable</b> commands. Refer to “Enable and Disable Commands” on page 4-62 for more information on the <b>enable</b> and <b>disable</b> 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

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 4-34 for a description of the bit fields in this register.

**Note:** PIO's with the bypass bit off may alter this register and therefore software should use a buffer number which is not being used (X'F' recommended).



- **Register 8 – Buffer Control**

This register contains a copy of the I/O bus address associated with the TCW register described in the preceeding 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:

<b>Bits</b>	<b>Description</b>
<b>0</b>	<b>Buffer Dirty:</b> This bit indicates that the buffer associated with this channel is <i>dirty</i> , that is, has been written to and therefore contains data which is inconsistent with data in system memory. This bit is reset by the IOCC when the buffer is flushed 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.
<b>1</b>	<b>Buffered:</b> This bit indicates that the buffer contains data which has been 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. When the operation completes and the device interrupts, the buffer must be flushed to system memory by software using the <b>buffer flush</b> command.
<b>2</b>	<b>Buffer Invalidate:</b> 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.
<b>3–5</b>	<b>Reserved:</b> These bits are reserved and must be set to a value of 0.
<b>6–25</b>	<b>I/O Bus Address A31 to A12:</b> 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 reaccesses the TCW table.
<b>26–31</b>	<b>I/O Bus Address A11 to A6:</b> 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 46 illustrates 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. Also not shown is the caching of the current TCW table entry. Figure 46 assumes direct access of the TCW table entry on each I/O access by the bus master.

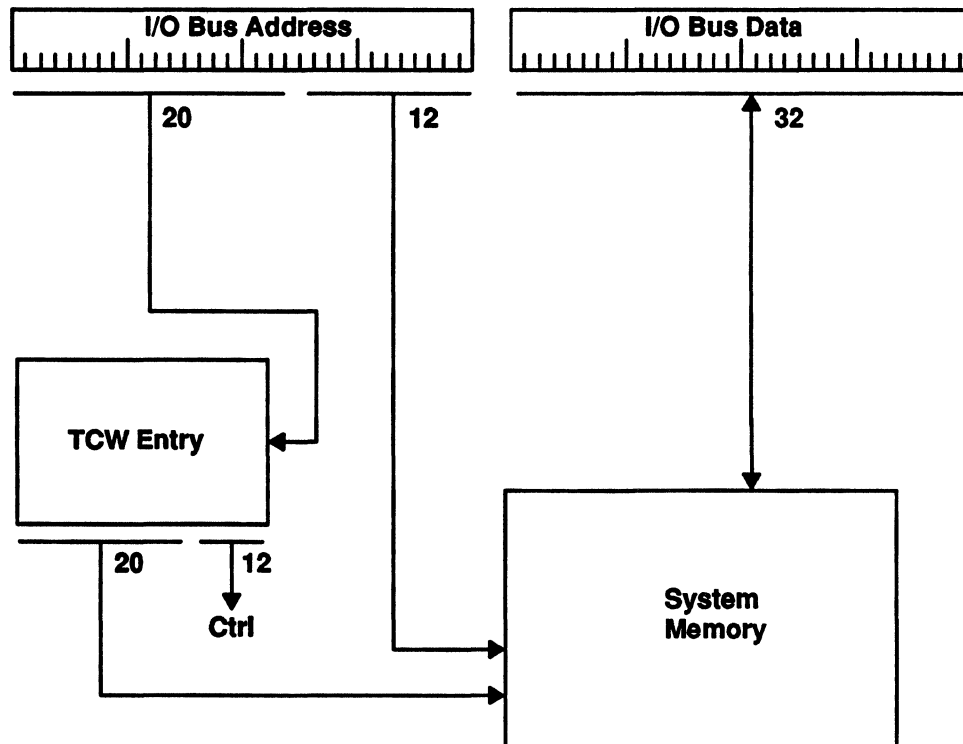


Figure 46. Unbuffered Bus Master Data Transfer Operation

The Bus Master Channel Status register for the unbuffered case is illustrated in Figure 47. 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.







Register fields are described below:

- Register 0 – Channel Status register

Bits	Description
0 – 3	Control and Status: For a description of these bits see “Buffered Bus Master ” on page 4-37.
4	DMA Slave Flag: For a description of this bit see “Buffered Bus Master” on page 4-37.
5–15	Reserved: These bits are reserved and must be set to a value of 0.
16–23	Authority Mask: For a description of these bits see “Buffered Bus Master ” on page 4-37.
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 illustrated in Figure 48, accesses are verified by checking the TCW memory protect key against an authority mask associated with the requesting channel.

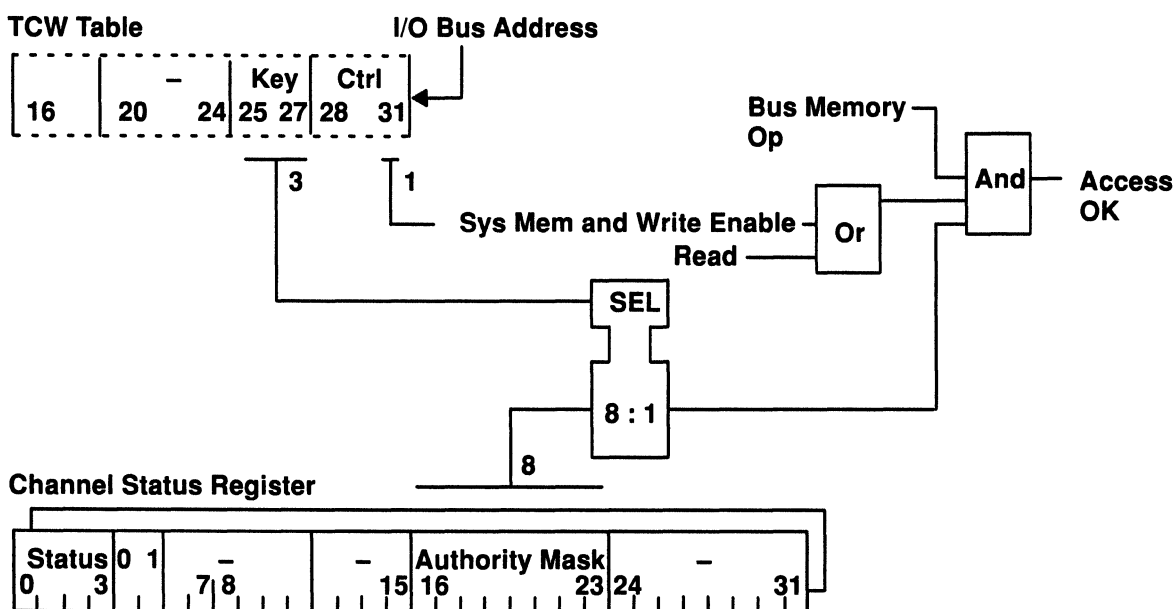


Figure 48. Bus Master Access Authority Checking

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, which 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 4-34. 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 4-45.

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 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, along with I/O bus address bits A31 to A6 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 'sfdbktrn' 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 4-80.

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 'sfdbktrn'. If the access is directed to the IOCC, the IOCC will 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 Error" on page 4-22 and "Channel Check" on page 4-22 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: 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: 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 Protocol Error: This error code is set if a Micro Channel protocol error has been detected (for example, the channel is mapped to system memory, but the bus master pulls the 'M/IO' line on the Micro Channel bus, indicating that it is doing an I/O operation).
1 0 0 1	Data Parity: 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 address is bad.
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 RISC System/6000 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" section on page 4-71 and "Implementation Details" on page 4-80").

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 TCW's and the other uses tag elements to handle this management of memory pages. See "DMA Slave Operations Using Tag's" on page 4-47 and "DMA Slave Operations Using TCW's" on page 4-52 for a description of these two modes. The choice of using TCW's or tag's for the management of the 4K-byte pages is implementation dependent (See "IOCC Configuration Register" section on page 4-71 and "Implementation Details" on page 4-80").



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 TCW's, the number of useable DMA channels is implementation dependent (see "IOCC Configuration Register" section on page 4-71 and "Implementation Details" on page 4-80").

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 uncorrected 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" section on page 4-62. 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 less 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 4-80). 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 policed 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 the "DMA Slave Error Conditions" on page 4-57 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" section on page 4-62 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 4-57 and “Enable and Disable Commands” on page 4-62.

The tags are organized as a heap in a special memory space called a tag table. The tag table includes 4096 entries and requires 32K bytes of memory. 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 49 illustrates DMA slave operations using tag elements. Data may be transferred between a device and system memory or between a device and bus memory. All data transfers to or from system memory have 64-byte granularity. In the buffered mode, the IOCC must provide a 64 byte data buffer 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 sort of read-modify-write capability so that transfers from the device, which will be less than 64 bytes, can be matched to the system memory interface. Data transfers to or from bus memory are not buffered.

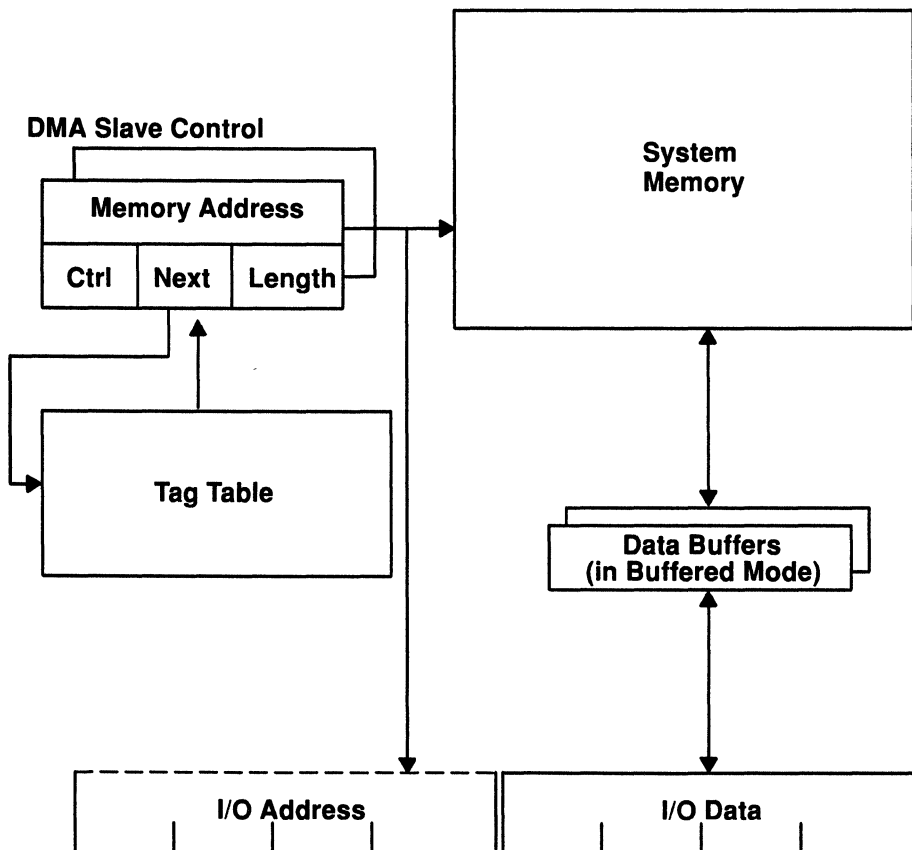


Figure 49. DMA Slave, Using Tag's



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 50 illustrates 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 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.

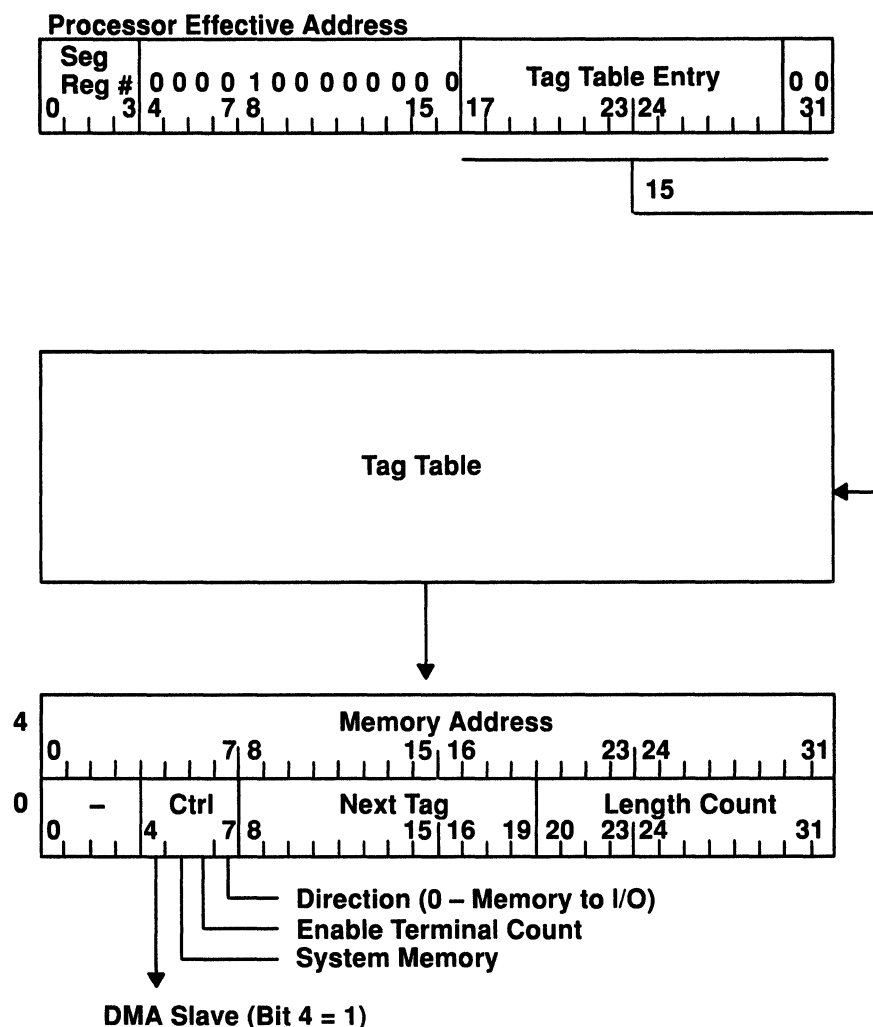


Figure 50. Tag Table Addressing

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 4 of a tag contains a 32–bit *real* address to either the bus memory space or system memory space.
- Word 0 of a tag contains control information relating to the current 4K–byte page and includes the following:



<b>Bits</b>	<b>Description</b>
<b>0–3</b>	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.
<b>4</b>	DMA Slave Flag: This bit is set to a value of 1 using an I/O Store instruction to personalize a DMA channel for a DMA slave operation. The IOCC never changes the value of this bit. The hardware does not update the Channel Status register bit 4 with this bit.
<b>5</b>	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.
<b>6</b>	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.
<b>7</b>	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.
<b>8–19</b>	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 1s indicates that this is the last tag in a chain. If this field is all 1s 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 1s and therefore cannot be used.
<b>20–31</b>	Length Count Field: This field contains a length count for the data transfer. This 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.

Figure 51 on page 4-51 illustrates 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.



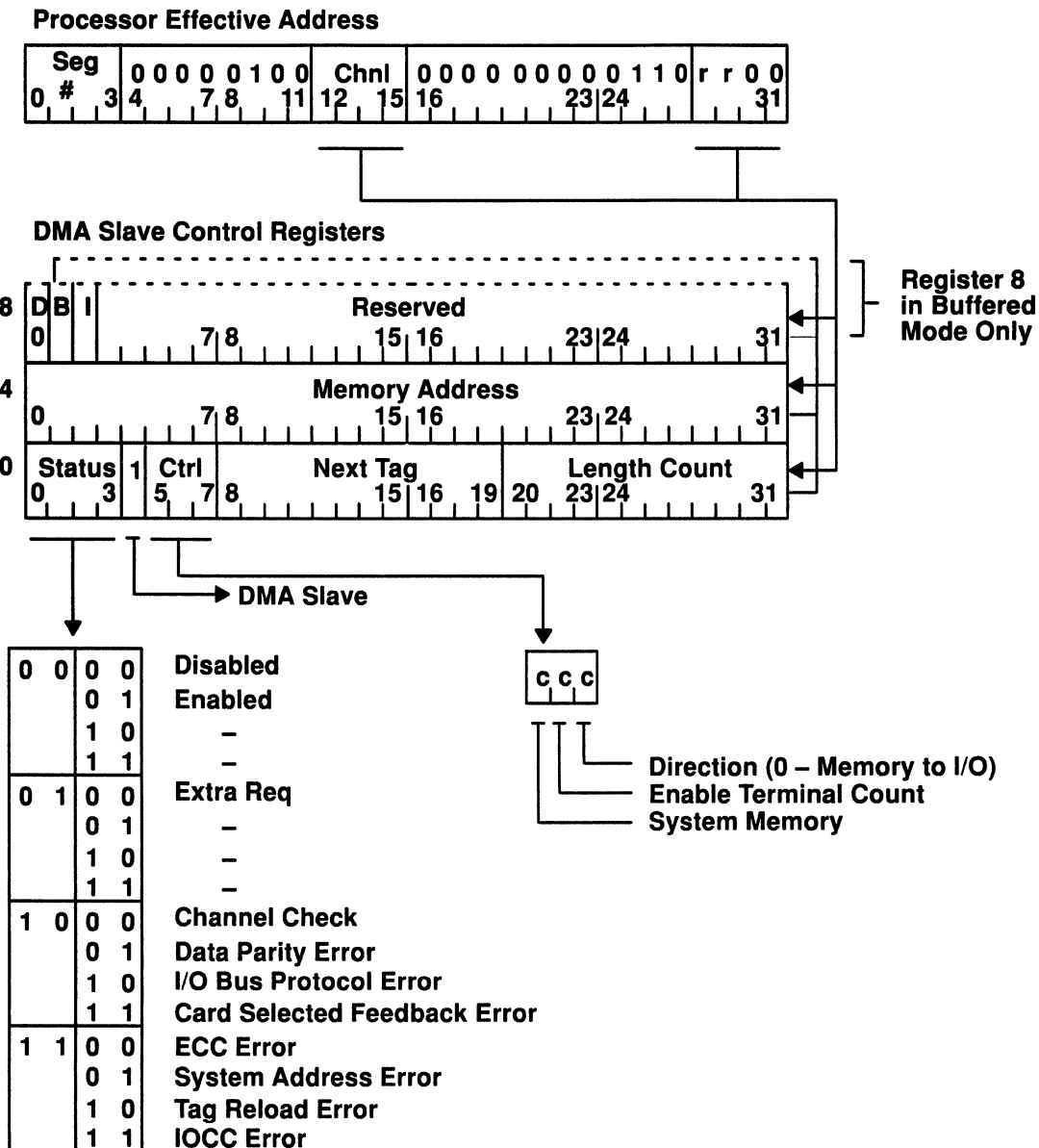


Figure 51. DMA Slave Registers Using Tag's

The register fields are as 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 <b>enable</b> command and reset using the channel <b>disable</b> 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 4-57 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 4-49.
- 5 System Memory Flag: This bit is defined the same as for the tag table word 0 defined on page 4-49.
- 6 Enable T/C Flag: This bit is defined the same as for the tag table word 0 defined on page 4-49.
- 7 Direction Flag: This bit is defined the same as for the tag table word 0 defined on page 4-49.
- 8–19 Next Tag Field: This bit is defined the same as for the tag table word 0 defined on page 4-49.
- 20–31 Length Count Field: This bit is defined the same as for the tag table word 0 defined on page 4-49.

- Register 4 – Memory Address Register

This register is defined the same as tag table word 4.

- 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 <i>dirty</i> , 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 has been 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. When the operation completes and the device interrupts, the buffer must be flushed to system memory by software using the <b>buffer flush</b> 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.
3–31	Reserved: These bits are reserved and must be set to a value of 0.

## DMA Slave Operations Using TCW's

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 52 on page 4-53 illustrates 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 4-34 for a description of the TCW table.

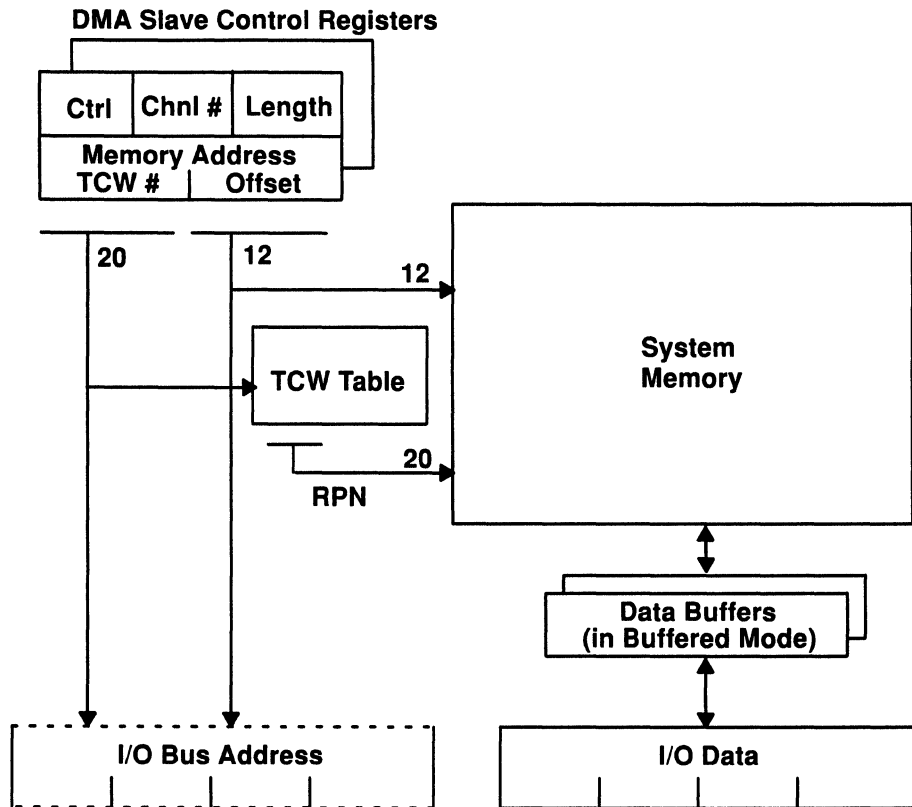


Figure 52. DMA Slave, Using TCW's

Figure 53 on page 4-54 illustrates the register definitions when TCW's are used to control DMA slave operation.



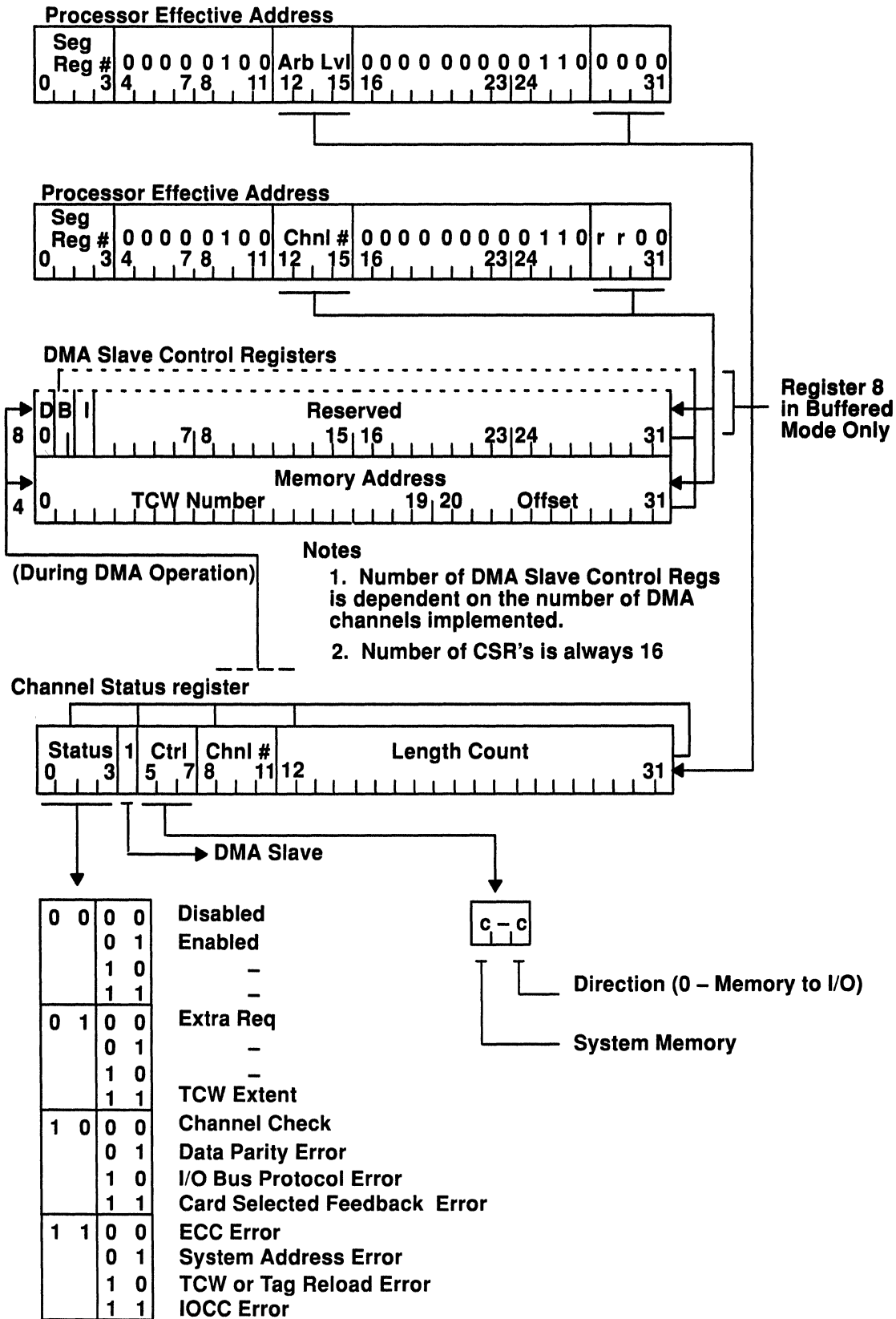


Figure 53. DMA Slave Registers, Using TCW's



The register fields are described below.

- 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 <b>enable</b> and reset using the <b>disable</b> 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 4-57 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 in "DMA Slave Operations Using Tags" on page 4-47. 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.
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 4-71 and "Implementation Details" on page 4-80). 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 a single channel number is never assigned to more than one CSR (arbitration level).

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 (RPN) 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 4-80). 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 “DMA Slave Operations Using Tags” on page 4-47.

## **DMA Slave Bus Protocols**

Conventional bus protocols are used in DMA operations and are documented in “Basic Transfer Cycle” on page 4-18.

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 4-21. 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 do 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 4-63 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 4-19.

## 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: 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.

<b>0 1 1 1</b>	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.
<b>1 0 0 0</b>	Channel Check: 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).
<b>1 0 0 1</b>	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 4-80 for details.)
<b>1 0 1 0</b>	I/O Bus Protocol Error: This error code is set if a Micro Channel Protocol error has been detected (for example, a card pulls the 'cd ds 32' line on the Micro Channel but does not pull the 'cd ds 16' line at the same time).
<b>1 0 1 1</b>	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.
<b>1 1 0 0</b>	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.
<b>1 1 0 1</b>	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.
<b>1 1 1 0</b>	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.
<b>1 1 1 1</b>	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. The Load or Store instruction can be either a string or non-string operation. Commands supported by the IOCC include:

- **Time delay**
- **End of interrupt**
- **Lock**
- **Enable and disable**
- **Buffer flush**
- **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 illustrated in Figure 54 on page 4-60. 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.



### Effective Address For Time Delay Command

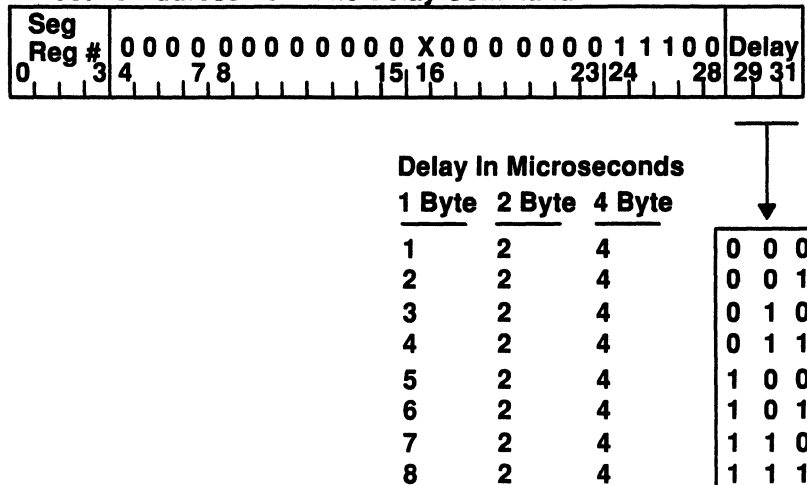


Figure 54. Time Delay Command

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 cannot be included as part of a string operation. 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

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 re-enables 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, illustrated in Figure 55, should be issued following the interrupt service.

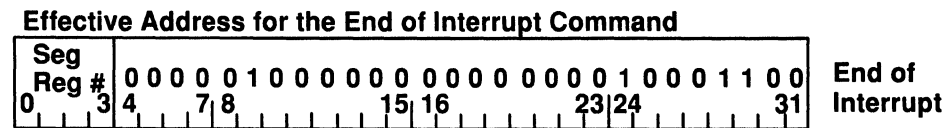


Figure 55. End of Interrupt

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.

## Lock Command

Devices on the Micro Channel I/O bus have the ability to momentarily suspend arbitration to guarantee the atomicity of sequential bus cycles. This is required when implementing functions such as a test and set or an exchange (jump) sequence. These require a read-modify-write sequence to be performed under one arbitration envelope and are characteristically used to control access to shared control areas.

The IOCC provides a **lock** prefix command for operating with devices dependent upon this use of shared variables. It causes the arbiter to suspend arbitration between two successive I/O Load and Store instructions, and should be placed prior to the first I/O instruction. Arbitration is suspended on the first I/O command following receipt of a **lock** command, even if it is directed to IOCC facilities. Thus, access to IOCC facilities such as the TCW tables, can be made atomic. The processor cannot be interrupted between the **lock** command and the second I/O instruction, and interrupts must be masked off during the atomic sequence.

Figure 56 illustrates command execution and the effective address format. It is permissible to insert test or logic instructions between the two I/O instructions. Although there are no restrictions on the number of instructions inserted, DMA and interrupt latency can be affected and it is recommended that reasonable caution be used. Any errors occurring during execution of the first I/O instruction cause the **lock** command to be canceled and arbitration re-enabled.



Effective Address For Lock Command

Seg Reg #	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
	3	4		7	8				15	16					23	24					31

Instruction Stream

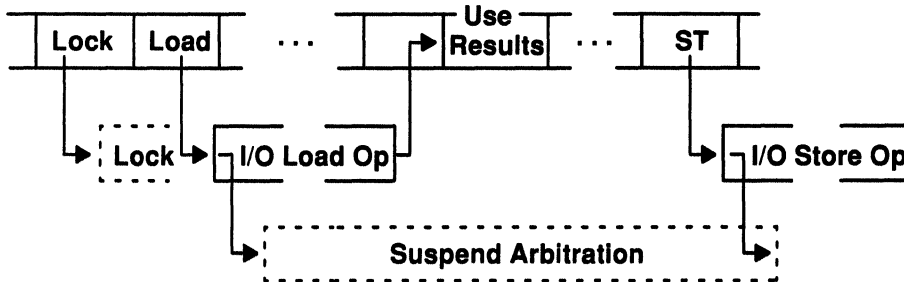


Figure 56. Lock Command

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 cause a Data Storage interrupt to be posted and invalid operation error status to be set in Channel Status register 15.

This instruction must be a Store instruction. The results of a Load instructions are implementation-dependent (see "Implementation Details" on page 4-80). On a Store instruction, the data is ignored.

**Note:** Use of this command requires extreme care. Failure to use it appropriately can cause a bus timeout. Therefore, the period of time between the first bus operation and the second must be absolutely guaranteed to be less than 7.8 microseconds. To ensure this, all interrupts must be disabled during the entire lock sequence, the lock must not cause an exception other than a bus exception, and must be restartable. Also, the kernels non-maskable interrupt handlers, such as machine check and system reset, must be programmed to abort the lock sequence and restart the entire lock sequence upon returning.

## 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 illustrated in Figure 57. 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

Seg Reg #	0	0	0	0	0	1	0	0	Chnl #	0	0	0	0	0	0	0	1	1	1	0	0	0	0
	3	4		7	8				12	15	16				23	24							31

Figure 57. 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 paragraph does not indicate a *do not care*, 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 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 will 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 will cause 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 which support IOCC buffers. 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 see "Implementation Details" on page 4-80.

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.

The **buffer flush** commands are protected system functions and may 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 CPU cache, and a flush operation is performed by the 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 58 illustrates 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 and B bits, the TCW number and the offset, and the invalidate bit (I) equal 1. This causes any subsequent accesses to this buffer to have to reaccess the TCWs and system memory. If on, the dirty bit is turned off, so any subsequent **flush** commands will not cause a buffer transfer.



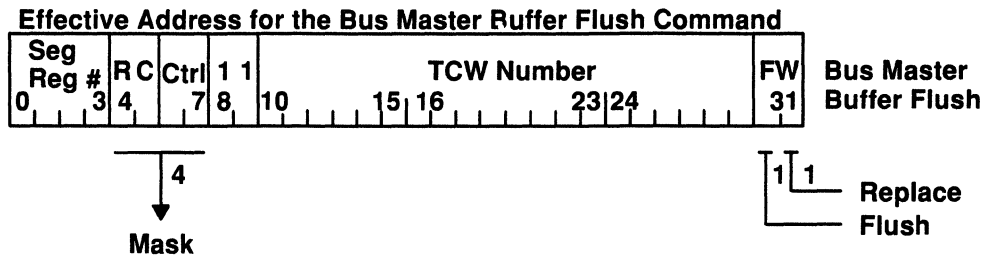


Figure 58. Bus Master Buffer Flush

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.

On a Load instruction, return the old value of the TCW. On a Store instruction, data is ignored. The TCW is updated based on the R, C, and CTL bits in the mask field.

- 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 buffer for the DMA slave is managed as simple buffers, and the flush operation is performed by channel number. The DMA Slave **buffer flush** command is illustrated in Figure 59 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.

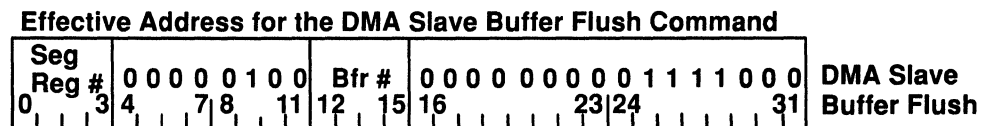


Figure 59. DMA Slave Buffer Flush

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. Following the data transfer, the dirty and buffered bits are reset and the invalidate bit (I) is set.

On a Store instruction, the data is ignored. A Load instruction causes a DSI. In the unbuffered mode, a Store instruction is a NOP and a Load instruction returns indeterminate data.



## Buffer Invalidate Command

Figure 60 illustrates the effective address format for this command.

### Effective Address for Invalidate Command

<b>Seg Reg #</b>	0 0 0 0	0 1 0 0	<b>Buf #</b>	0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0
0   3	4   7	8	15   16	23   24   31

### Figure 60. 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. On a Store instruction, the data must be X'20000000'.

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.

## 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 (IRQ's) to the EIS register. The specific approach supported is implementation dependent (See "Implementation Details" on page 4-80). 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 61 illustrates the interrupt mechanism.



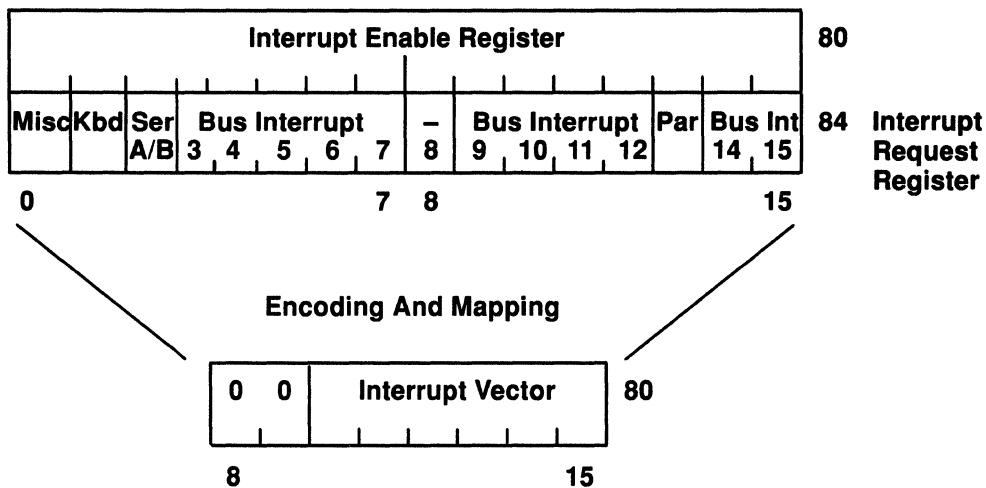
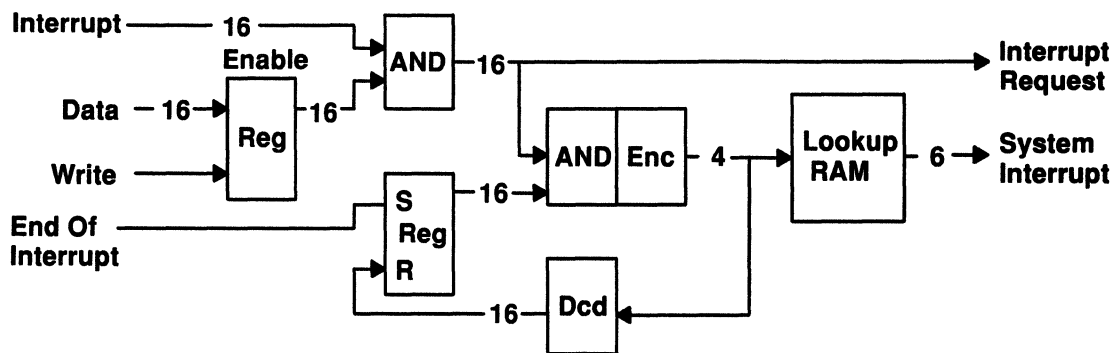


Figure 61. Interrupt Mechanism

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 the "Processor Architecture" section of *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — General Information 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 illustrated in Figure 62 on page 4-67. 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







- **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:

<b>Bits</b>	<b>Description</b>
<b>0</b>	Miscellaneous Interrupt: miscellaneous interrupts are not directly vectored to the EIS register. The RISC System/6000 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.
<b>1</b>	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.
<b>2</b>	Serial Port Interrupts: This bit is set to a value of 1 when a board serial port A or serial port B 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.
<b>3–7, 9–12, 14–15</b>	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.
<b>8</b>	Reserved: This bit is reserved and must be set to a value of 0.
<b>13</b>	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.
<b>16–31</b>	Reserved: These bits are reserved and must be set to a value of 0.



- **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:

Bit	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 RISC System/6000 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 4-80 and "Channel Check" on page 4-22 for more information.
1	Bus Timeout: This bit is set if an I/O bus timeout occurred. See "Bus Timeout" on page 4-22 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 4-80).
3–31	Reserved: These bits are reserved and must be set to a value of 0.

- **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 4-80). Implementations that support a single I/O bus can fix the conversion of interrupt level to the EIS bit. This fixed conversion will be 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 63 illustrates the register organization within the IOCC. (For implementation details, see "Implementation Details" on page 4-80.)

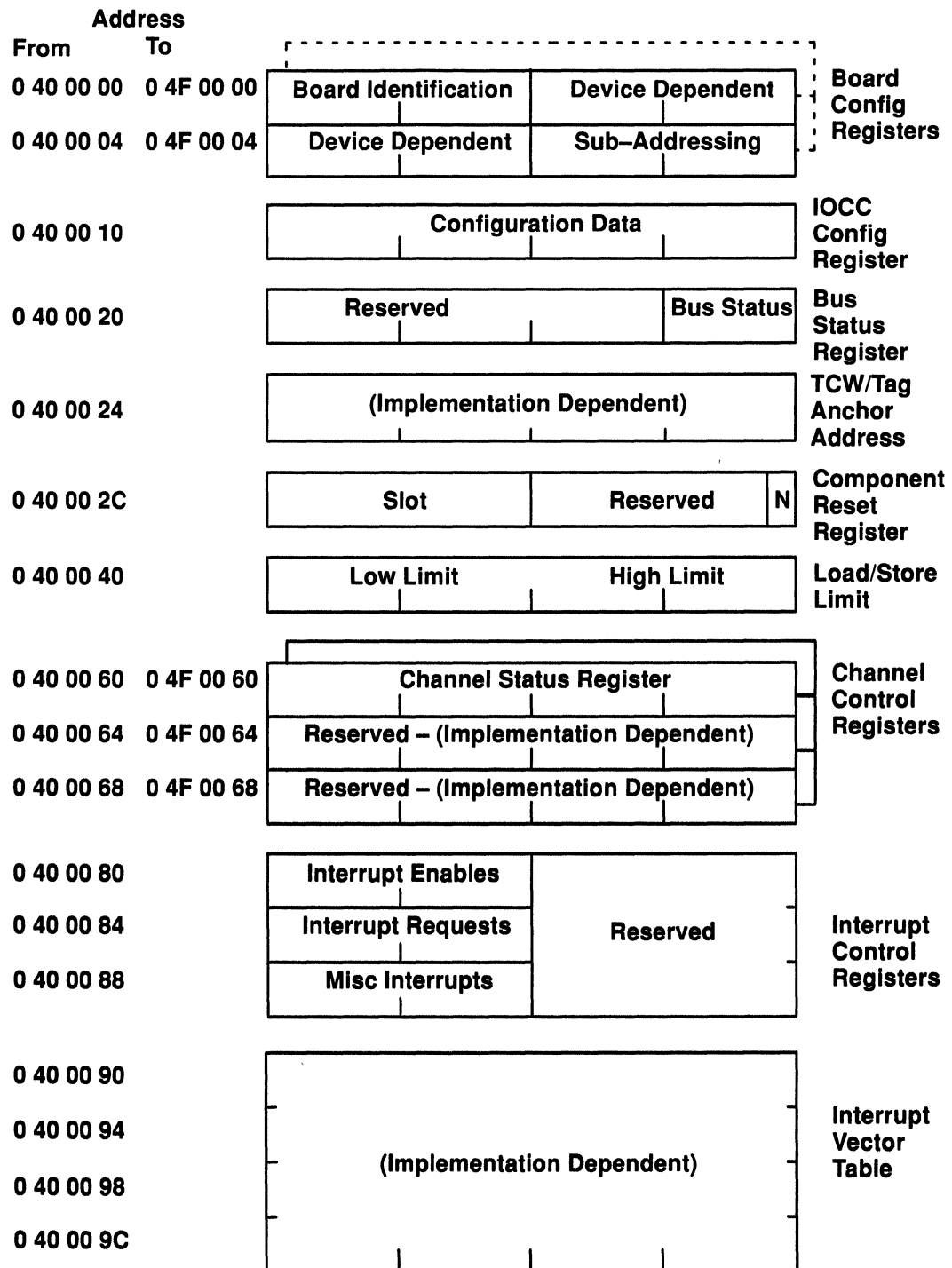


Figure 63. IOCC Registers



## Board Configuration Data

The Micro Channel defines a slot select mechanism for accessing board-unique configuration data (byte-only access). Eight bytes of addressing is 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 illustrated in Figure 64. 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.

### Processor Effective Address

Seg Reg #	0	0	0	0	0	1	0	0	Slot	0	0	0	0	0	0	X	0	0	0	0	r	r	r
	0	3	4	7	8				15	16						23	24					31	

### Data

Board Identification LS Byte (X0)								Dev Unique				E	Dev Unique								0
												N									
Dev Unique				Sta	Dev Unique				Sub-Addressing LS Byte								MS Byte (X7)				4
0	3	4	7	8					15	16			23	24							31

Figure 64. Board Configuration Registers

Refer to the *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture* manual for 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 are variables 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 64.

## 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 4-80.

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

0x400010



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 65 illustrates the organization of the configuration register. Bit 0 in this register is initialized to a value of 0 at startup.

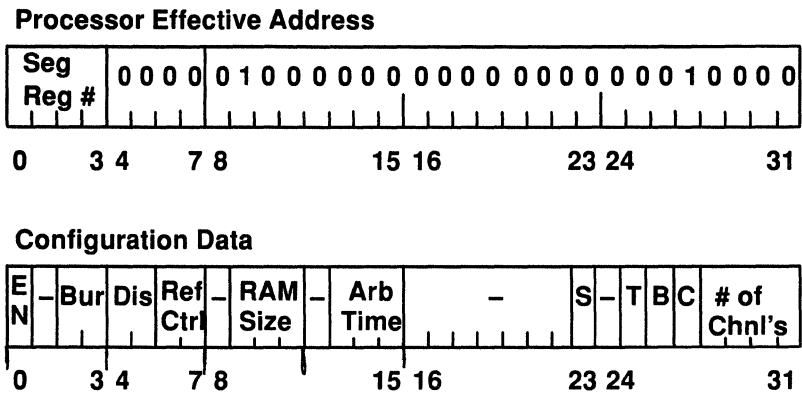


Figure 65. IOCC Configuration Register

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 4-80). 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 66. 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** Disable Control: These two bits are implementation dependent (see “Implementation Details” on page 4-80).

**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 follows (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 67. Refresh Control Setting

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

**9–11** RAM Size Specification: These bits allow specification of the amount of control RAM 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. The following table shows the net sizes of the TCW table and Tag table resulting for each size provided:

Bit				
9	10	11	RAM Size	TCW Table
0	0	0	128K–byte	96K–byte
0	0	1	256K–byte	224K–byte
0	1	0	512K–byte	480K–byte
0	1	1	1M–byte	992K–byte
1	0	0	2M–byte	2016K–byte
1	0	1	4M–byte	4064K–byte
				32K–byte

**Note:** Tags used for DMA slave operation (Bit 25 equals 0).

Figure 68. RAM Size Specification for Combination TCW and Tag Table



Bit	RAM Size		TCW Table
9 10 11			
0 0 0			32K-byte
0 0 1			64K-byte
0 1 0			128K-byte
0 1 1			256K-byte
1 0 0			512K-byte
1 0 1			1M-byte
1 1 0			2M-byte
1 1 1			4M-byte

**Note:** TCWs used for both DMA slave and bus master operation (Bit 25 equals 1).

Figure 69. RAM Size Specification for TCW Table

The Tag table has 32K bytes, and the remainder 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, there is no bus memory allowed between 0K 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.

**13–15**

**Arbitration Time:** These bits allow specification of the arbitration time on the Micro Channel I/O bus. 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 table represents a range, for example, 100 nanoseconds equals 100 to 200 nanoseconds.

Bits	Arbitration Time
13 – 15	(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

Figure 70. Arbitration Time Configurations

**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'0400024'.
- 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 or tag access error. This area is not scrubbed.
- A value of 0 in this bit indicates that non-system memory is used for the TCW and tag tables.
- 24**      **Reserved:** This bit is reserved and must be set to a value of 0.
- 25**      **DMA Slave TCW and 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.
- 26–27**    **Cache Buffer Support and Cache Coherency:** These bits have the following meanings:

26	27	
0	0	<b>Buffered Mode, Software Enforced Consistency</b>
0	1	<b>Unbuffered Mode</b>
1	0	<b>Reserved</b>
1	1	<b>Reserved</b>

Figure 71. 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.

- 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 4-80). If the implementation supports tag's, then all 15 DMA slave channels must be supported. The minimum required by the Micro Channel architecture is 2. The minimum required by the RISC System/6000 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 72 illustrates the Bus Status register .

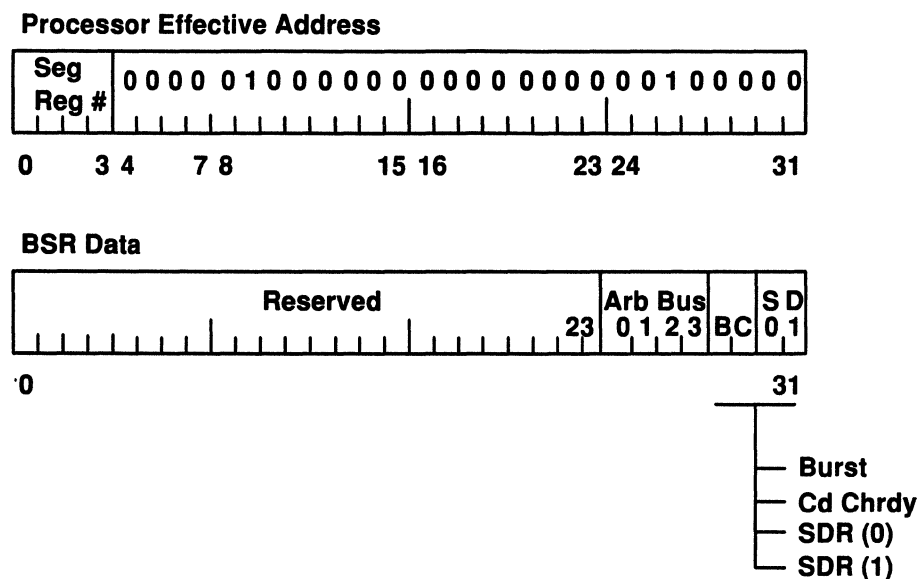


Figure 72. Bus Status Register

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 4-80). 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.

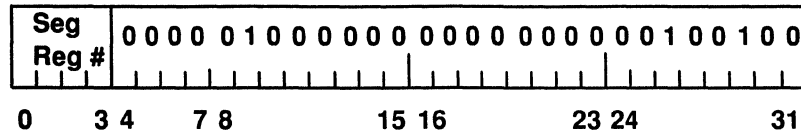
## TCW/Tag Anchor Address Register

This register specifies the starting address of the TCW/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/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 these registers 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 73 on page 4-77 illustrates the TCW/Tag Anchor Address register.



#### Processor Effective Address



#### Anchor Address Register Data

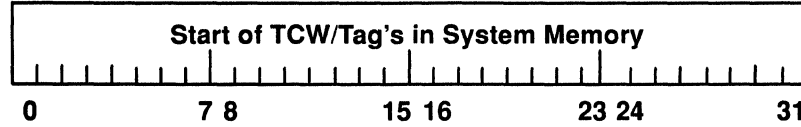


Figure 73. TCW/Tag Anchor Address Register

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.

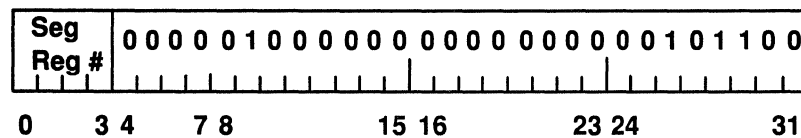
### Component Reset Register

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. All Standard I/O adapters are reset by one 'reset' signal controlled by bit position 31 in the CRR.

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 74 shows the Component Reset register. The actual number of slots supported is implementation dependent (See "Implementation Details" on page 4-80) and is consistent with the IOCC configuration definition. On a Load instruction to this register, the value of bits 16 to 30 and the unused bits in the slots field are implementation dependent.

#### Processor Effective Address



#### Component Reset Register Data

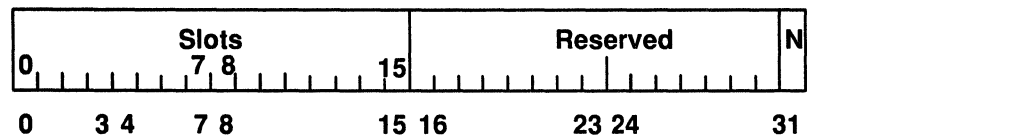


Figure 74. Component Reset Register

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.

---

## System I/O and Standard I/O

Two classes of devices are described in this section, the System I/O and 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 RISC System/6000 unit are defined as those I/O devices intrinsic to a basic workstation, and as such, are included as part of the base machine. Not being optional features, these devices do not necessarily occupy feature slots. The list of items which fall into this category is implementation specific (see "Implementation Details" on page 4-80).

### 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 FC' defining a contiguous space of 64 bytes. These registers are implementation-dependent (See "Implementation Details" on page 4-80).

### 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 2 M-bytes of address space. The amount of NVRAM in the system is implementation-specific (see "Implementation Details" on page 4-80).

### 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. I/O addresses between X'00 00' and X'00 FF' are reserved for the Standard I/O. Figure 75 on page 4-79 illustrates the Standard I/O addressing.

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 4-80).



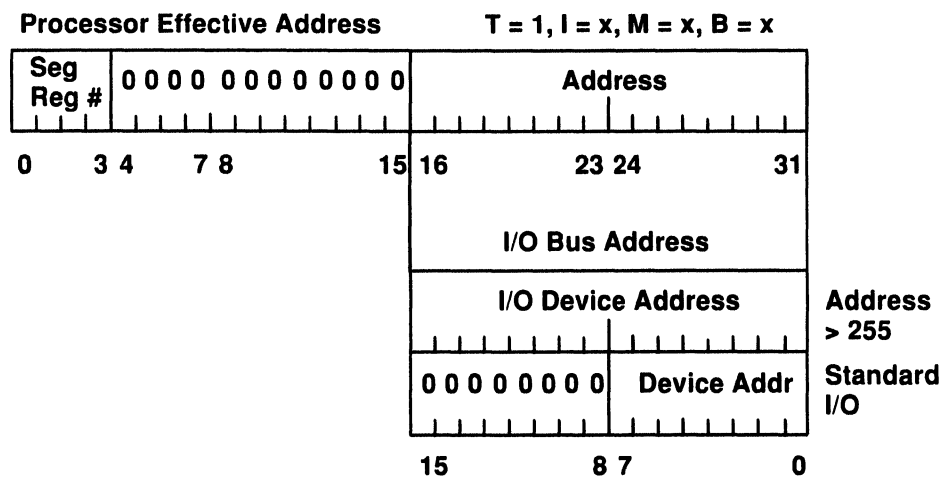


Figure 75. Standard I/O Addressing



---

## Exception Reporting and Handling

The *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture* manual contains a section entitled "Exception Condition Reporting and Handling" that defines the data and address parity on the Micro Channel.

The following are general guidelines that were followed in designing the RISC System/6000 units and adapters, and should be followed in designing new adapter boards for the RISC System/6000 machines:

- Full parity support is recommended for all address and data buses for all RISC System/6000 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 busses (generate parity at the signal source and check parity at each destination point where the address and data will be used).
  - Internal RISC System/6000 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 RISC System/6000 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 resistors 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. 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 "Exception Condition Reporting and Handling" section of the *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture*.

---

## Implementation Details

This section provides specific implementation details for all RISC System/6000 units.

### IOCC Configuration Register

Some of the bits in the IOCC Configuration register indicate support or non-support of various implementation dependent features. The following is a summary of the definition of the RISC System/6000 IOCC Configuration register implementation. In the case of read only memory (ROM) code initialized bits, the value that the ROM must initialize these bits to is shown.



<b>Bits</b>	<b>Description</b>
<b>2–3</b>	Burst Control: RISC System/6000 units support the use of the programmable burst control as indicated in bits 2 and 3 of the IOCC Configuration register. These bits are set to B'11' by the ROM code.
<b>4–5</b>	Reserved: These bits are reserved and must be set to B'01'.
<b>6–7</b>	Refresh Control: These bits are set to B'01' (60 microsecond refresh) by the ROM code.
<b>9–11</b>	RAM Size Specification: These bits are set to B'010' by the ROM code.
<b>13–15</b>	Arbitration Time: These bits are set to B'011' (400 nanoseconds) by the ROM code.
<b>23</b>	TCW/Tag Tables in System Memory: RISC System/6000 units support the use of non–system memory for TCW and tag tables as indicated by a 0 in this bit.
<b>25</b>	DMA Slave TCW/Tag: RISC System/6000 units support the use of tags for DMA slave operations as indicated by a 0 in this bit..
<b>26–27</b>	Buffer Support/Coherency: RISC System/6000 units 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 RISC System/6000 units support PIO operations to system memory.
<b>28–31</b>	Number of DMA Slave Channels: RISC System/6000 units support the use of 15 channels for DMA slave operations as indicated by B'0000' in these bits.

## System Registers

Figure 76 on page 4-82 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 in the RISC System/6000 units is a software function).



Address	Data	System Registers
0 40 00 C0	Time of Day Clock and Alarm	
0 40 00 C4	Time of Day Clock and Alarm	
0 40 00 C8	Time of Day Clock and Alarm	
0 40 00 CC	Time of Day Clock and Alarm	
0 40 00 D0	Time of Day Clock and Alarm	
0 40 00 D4	Time of Day Clock and Alarm	
0 40 00 D8	Time of Day Clock and Alarm	
0 40 00 DC	Time of Day Clock and Alarm	
0 40 00 E0	System Reset Count	
0 40 00 E4	Power Status and Keylock Decode	
0 40 00 E8	Power Control and Reset	
0 40 00 EC	Diagnostic Control	
0 40 00 F0	Reserved	
0 40 00 F4	Reserved	
0 40 00 F8	Reserved	
0 40 00 FC	I/O Board Part No and EC Level	

Figure 76. System Registers

## Nonvolatile RAM

For the RISC System/6000 units, only 32K bytes of nonvolatile random access memory (NVRAM) is presently planned and is located in the lower 32K-byte area of this space. Figure 77 on page 4-83 illustrates the address assignments for the NVRAM area.



Address	Data	
0 A0 00 00 (4 Bytes)	Reserved	Protected Software or ROM Access Only
0 A0 00 04 (4 Bytes)	NVRAM Size	
0 A0 00 08 (4 Bytes)	Date and Time NVRAM Initialized	
0 A0 00 0C (4 Bytes)	Reserved	
0 A0 00 10 (4 Bytes)	SCSI Initiator Address Slot 1–16	
0 A0 00 14 (4 Bytes)	Reserved	
0 A0 00 18 (4 Bytes)	Reserved	
0 A0 00 1C (4 Bytes)	Reserved	
0 A0 00 20 (224 Bytes)	Memory Control And Error Registers Mapped From BUID 0 Address 1000–10D0	
0 A0 01 00 (256 Bytes)	Memory Error Summary Data	
0 A0 02 00 (36 Bytes)	Previous IPL Device Descriptor	Hardware Prevents OCS Write To This Area
0 A0 02 FC (4 Bytes)	Software CRC Value For A0 00 00 – A0 02 FB	
0 A0 03 00 (4 Bytes)	LEDs (Mirrored)	
0 A0 03 04 (4 Bytes)	LEDs (Mirrored)	
0 A0 03 08 (4 Bytes)	Check Stop Count	
0 A0 03 0C (4 Bytes)	PTR To OCS Logout Area Lt 00 A0 44 00	
0 A0 03 10 (4 Bytes)	OCS Code EC Level	
0 A0 03 14 (4 Bytes)	Seeds ROM, EC Level	
0 A0 03 18 (4 Bytes)	Manufacturing Control Word	
0 A0 03 1C (4 Bytes)	Pointer To Manufacturing Data Area	
0 A0 03 20 (64 Bytes)	OCS LED String Output Area	Shared Access OCS, Software, ROM
0 A0 03 60 (4 Bytes)	Pointer to OCS Code Exec. Area	
0 A0 03 64 (4 Bytes)	Pointer to OCS Work Area	
0 A0 03 68 (20 Bytes)	Machine Check Error Save	
0 A0 03 7C (4 Bytes)	OCS and RS Command Interface	
0 A0 03 80 (128 Bytes)	Reserved for OCS Buffer to RS Proc.	
0 A0 04 00 (16K Bytes)	OCS Work and Code Area	
0 A0 44 00 (15,360 Bytes)	Software Data Area	
		OCS Area
		Software Area

Figure 77. NVRAM Addressing



## Standard I/O

Figure 78 is a Standard I/O address map indicating the address assignments for each Standard I/O device.

Hex Address Range	Standard I/O Device
0000 – 002F	Reserved
0030 – 0037	Serial Port 1
0038 – 003F	Serial Port 2
0040 – 0041	Serial DMA Registers
0042 – 0047	Reserved
0048 – 004F	Mouse
0050 – 0059	Keyboard/Tablet/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

Figure 78. Standard I/O Address Map

## Bus Master Transfers

Bus master operations follow the buffered mode of operation (see “Buffered Bus Master” on page 4-37).

## Component Reset Register

The RISC System/6000 units support eight slots plus the Standard I/O. Bits 0–7 of this register represents the eight slots and bit 31 is for the Standard I/O. On a Load instruction, the value of bits 8 to 30 will be indeterminate.

## 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 4-80) 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 RISC System/6000 units support the coded method of handling I/O interrupts as described in the architecture including the use of the interrupt vector tables.



## Lock Command

On a Load instruction, a Data Storage Interrupt will result.

## Power-On Reset

A power-on reset, system reset, or check stop condition resets the master enable bit in the Configuration register. When this bit is a value of 0, the following is accomplished:

- The Component Reset register is reset
- A reset condition is forced to all I/O slots
- The 'preempt' signal is de-gated, disabling channel arbitration
- Interrupt presentation is inhibited to the system.

This register bit can be set or reset by a Store instruction to the IOCC Configuration register. Figure 79 illustrates the system implementation.

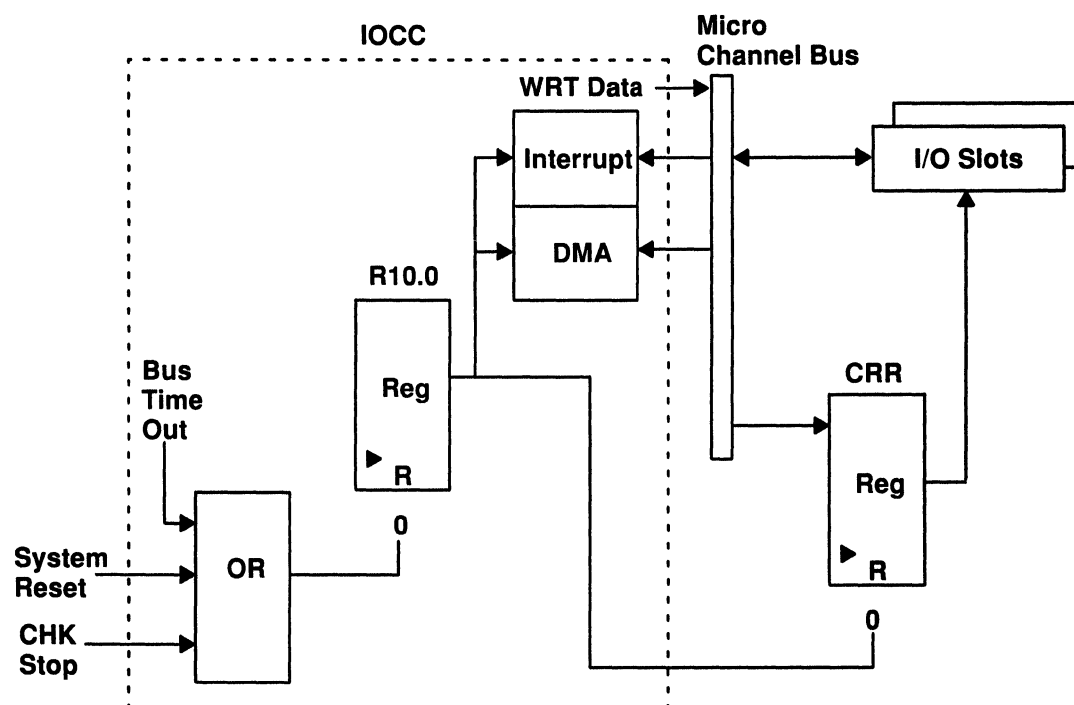


Figure 79. System Reset

## IPL Procedures

Figure 80 on page 4-87 illustrates the power-on state of the IOCC registers. Indeterminate power-on states are indicated with an *x*, and undefined states are indicated with a *-*. 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 Channel Control registers, register 8 bit 2(l) is a value of 1, all other bits are 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.



Address						
From	To					
0 40 00 00	0 4F 00 00	Board Identification		Device Dependent		Board Config Registers
0 40 00 04	0 4F 00 04	Device Dependent				
0 40 00 10		0xxx xxxx	xxxx xxxx	—		IOCC Config Register
0 40 00 20		—			0000 0000	Bus Status Register
0 40 00 2C		0000 0000			0	Component Reset Register
0 40 00 40		xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	Load and Store Limit
0 40 00 60	0 4F 00 60	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	Channel Control Registers
0 40 00 64	0 4F 00 64	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	
0 40 00 68	0 4F 00 68	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	
0 40 00 80		xxxx xxxx	xxxx xxxx	—		Interrupt Control Registers
0 40 00 84		xxxx xxxx	xxxx xxxx			
0 40 00 88		xxxx xxxx	xxxx xxxx			
0 40 00 90		xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	Interrupt Vector Table
0 40 00 94		xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	
0 40 00 98		xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	
0 40 00 9C		xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	

Figure 80. IOCC Power-On States



## Architectural Deviations

The following are implementation specific deviations from the I/O architecture.

- On a Load or Store error with the Bypass bit off, the error address bits A31 to A6 are not put into Channel Status register 15. However, since this error causes a Data Storage interrupt, the processor chip set saves the address of the failing instruction in the Machine Status Save and Restore Register 0 (SDR 0), and that address can be used to determine where the failure occurred.
- On a Load or Store, 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 which 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 operation, 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.
- Streaming data is not supported for IOCC initiated Load or Store, and DMA Slave operations.



---

## Chapter 5. Vital Product Data

### Chapter Contents

Description .....	5-3
Importance .....	5-3
Characteristics .....	5-3
Customer and Service Personnel Assistance .....	5-3
VPD Structural Overview .....	5-4
System Data Set .....	5-5
Keyword Descriptor Summary .....	5-5
Descriptor Keywords .....	5-5
Hardware VPD Descriptor Summary .....	5-10
Rack Record .....	5-10
Implementation Notes .....	5-10
Enclosure Record .....	5-10
Implementation Notes .....	5-10
Processor Board Record .....	5-11
Implementation Notes .....	5-11
I/O Board Records .....	5-11
Implementation Notes .....	5-11
Memory Records .....	5-11
Implementation Notes .....	5-11
Extra I/O Board Record .....	5-12
Direct Access Storage Device (DASD) Records .....	5-12
Device Required Descriptors .....	5-12
Optional Descriptors .....	5-12
Micro Channel Adapter Requirements .....	5-13
Preferred Implementation – POS Configuration Registers .....	5-14
System Configuration Protocol .....	5-15
Extended Storage Facility .....	5-16
Sample Layout of the Micro Channel Adapter VPD .....	5-17







---

## 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.

Certain information such as machine type, model and external serial number, for example, deskside system numbers, are 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 AIX 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:

1. Assists the operating system in auto-configuring the system and its components.
2. Assists diagnostics in problem determination and fault isolation:
  - a. Error logging includes VPD information so that a historical entry is associated with a serialized unit (such as an adapter).
  - b. Identifying the physical and logical location of failing units for replacement.
3. Assists the operating system in determining the proper device driver and loadable microcode level.
4. Assists the user in maintaining asset and inventory control.
5. Provides a means of licensing software on a processor ID or serial number basis.

## Characteristics

Configuration and VPD have the following characteristics:

- Vital product data is available at the rack, drawer, and field replaceable unit (FRU) level.
- For compatibility verification and testing, pluggable FRUs or potentially pluggable FRUs are required 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 may 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 RISC System/6000 system, for example, adding a DASD drawer to a rack system, the user or service personnel is required to enter information regarding its physical location and properties using the System Management Interface Tools.



## VPD Structural Overview

A system-level file or data set contains the fully expanded information on all vital product data elements for each enclosure component. The tree structure so formed begins with a rack or an enclosure level and goes on to identify all system components logically connected.

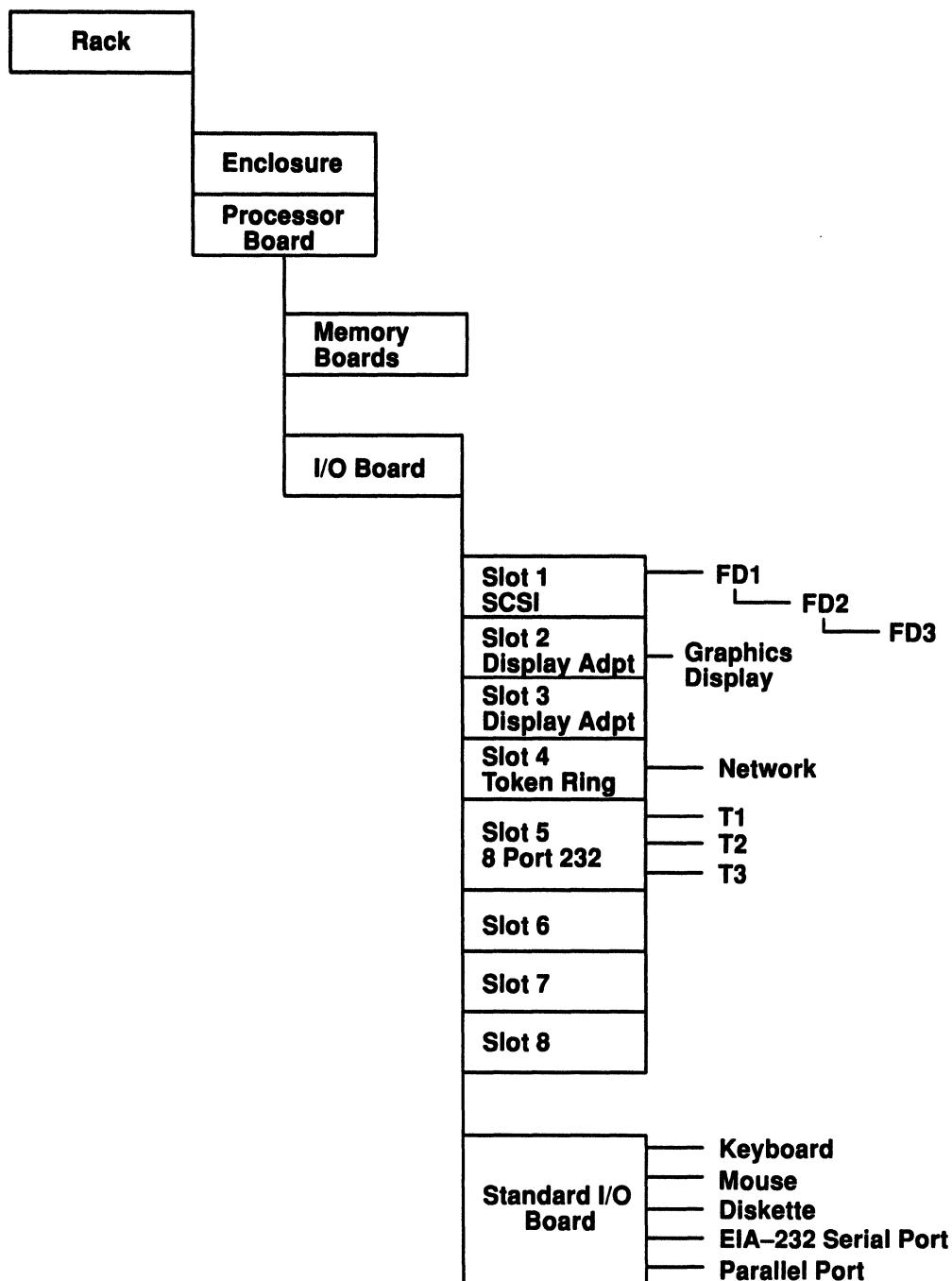


Figure 81. Configuration Tree



## 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 vital product data 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. 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 left-justified padding, as required.

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.

## Descriptor Keywords

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 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 RISC System/6000 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. (POS stands for Programmable Option Select, which replaces switches on feature boards. It is defined under the section "Micro Channel Adapter Requirements" on page 5-13.) 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.

- **\*DC L = date code**

The data is in ASCII character format.

- **\*DD L = device driver level (minimum required)**

The data portion of this descriptor is in ASCII. 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. This field is required for all adapters. 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. 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. This field is required for all adapters. 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 and specifies a drawer location in Electronics Industries Association units. It represents the drawer location within a rack for an enclosure. Environmental Impact Association (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)**

This is an optional field that may include a message to be printed or displayed for this record type. The ASCII character \* should be avoided within the data content of this message.

Micro Channel adapters designed for the RISC System/6000 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.



- **\*EC L = engineering change level**

The data portion of this descriptor is in ASCII. The characters are alphanumeric and represent the engineering change level for this element. The values of L, which range from 3 to 8, represent descriptor data counts of 2 to 12 alphanumeric characters. This descriptor number is right-justified and may be padded with high-order zeros or blanks for display or printout.

- **\*FC L = feature code**

The feature code is equivalent to the information contained in the \*TM L keyword (machine type and model). For example, due to the ordering system being used, the primary rack in a rack system is the machine type and model. Secondary racks that may be attached are designated as feature codes, not machine types. Therefore, this keyword is used to describe the feature code by which a system component is ordered.

The data is in ASCII character format.

- **\*FN L = FRU number**

The data portion of this descriptor is in ASCII. The characters are alphanumeric and represent the IBM FRU for this element of the RISC System/6000 product. Currently, this descriptor is an 8-position field.

- **\*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 sub-address facility or any other method to implement loadable microcode.

The data portion of this descriptor is an address pointer in the POS sub-address 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. 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 may 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 has been 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 may 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 IBM-built components, the first three characters are IBM. The next three characters are alphanumeric and are a code assigned to each IBM 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 IBM location establishing the contract is concatenated to the purchase order number.

- **\*NA L = network address**

This keyword is used by adapters that require a unique network address for a local area network. Adapters such as token-ring, or baseband use this field. This descriptor varies in size and data type as specified by an individual adapter.

- **\*NX L = pointer to next adapter VPD for multi-board adapters**

This is an optional field used by multi-board adapters that occupy more than one board slot on an I/O bus. The first board encountered in a slot position must provide POS registers. Additional adapter boards must be plugged into the next higher slot position. If the next adapter does not implement POS registers, this field specifies the VPD address to be specified in POS registers 7 and 6, respectively, in order to access VPD data on a second, third, or fourth additional adapter. The field is two bytes in binary format, and is accessed using the POS 3 register or port 3 in the same manner as for the first adapter board.

- **\*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. The characters are alphanumeric and represent the IBM part number for this element. The values of L, which range from 3 to 8, represent descriptor data counts of 2 to 12 alphanumeric characters. This descriptor number is right-justified and may be padded with high-order zeros for display or printout.

- **\*RA L = pointer to ROM code on adapter**

This is an optional descriptor type available for use. If an adapter chooses to access on-board ROM using the POS registers for reading microcode, this field is required. Adapters can use the POS extended-addressing facility or any other method to implement access to ROM.

The data portion of this descriptor is an address pointer in the POS sub-address space. Byte 0 is the most significant address byte, and byte 1 is the least significant address byte in binary.

- **\*RL L = ROM level and ID**

The data portion of this descriptor is in ASCII and is a minimum of four characters. Optionally, a second field of information of variable length specifies the ROM ID. This



additional information is required if more than one ROM is located on an individual adapter or board.

- **\*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/Write registers using POS registers, this field is required. Adapters can use the POS sub-address space or any other method to implement access to Read/Write registers and storage.

The data portion of this descriptor is an address pointer in the POS sub-address space. Byte 0 is the most significant address byte, and byte 1 is the least significant address byte in binary-form.

- **\*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 serial number is specified as an even number of ASCII alphanumeric characters in the range from 00000000 to ZZZZZZZZ with 8 characters as the maximum size. The number is right-justified and software may extend the high-order positions with zeros for display or printing.

- **\*SZ L = size**

Memory board adapters use this description to specify the size in M bytes. The data portion contains 2 ASCII numbers representing the useable memory configured with this adapter.

- **\*TM L = machine type and model**

The data portion of this descriptor specifies the machine type in ASCII in a length of 4 characters, followed by a machine model of 4 characters, for a total data length of 8 characters. Therefore, L is specified as 6 representing 8 characters of data plus the header.

- **\*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 sub-address 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 non-contiguous keyword descriptor data.

The data portion of this descriptor is an address pointer in the POS sub-address space. Byte 0 is the most significant address byte, and byte 1 is the least significant address byte in binary-form.



- \*Z0 – \*ZZ L = available for adapter-specific use.

---

## Hardware VPD Descriptor Summary

The following sections define the minimum requirements of various hardware components of a system.

### Rack Record

The required descriptors for the rack record are as follows:

Keyword	Description
*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

The required descriptors for the enclosure record are as follows:

Keyword	Description
*SN L	Serial number (externally visible)
*TM L	Machine type and model or *FC L = feature code
*PI L	Processor ID (from IPL ROM – for license requirements)
*DL L	Drawer level (if rack-mounted)
*MF L	Manufacturer.

### Implementation Notes

An enclosure represents a physical package. It may 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

The required descriptors for the processor board record are as follows:

Keyword	Description
*PN L	Board part number
*EC L	EC level
*RL L	ROM level and ID (IPL ROM)
*RL L	ROM level and ID (OCS ROM)
*RL L	ROM level and ID (seeds ROM)
*PC L	Processor component definition (specifies speed and processor model).
*Z0 L–*Z9L	Processor chip 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 chips.

## I/O Board Records

The required descriptors for the I/O Board records are as follows:

Keyword	Description
*EC L	EC 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, 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

The required descriptors for the memory records are as follows:

Keyword	Description
*EC L	EC level
*SL L	Slot location (software)
*Z0 L	EC level Left Data Multiplexer (Mux) chip
*Z1 L	EC level Right Data Mux chip
*Z2 L	EC level Controller chip.

### 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.

## Direct Access Storage Device (DASD) Records

The exact information may 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 disk enclosure unit as well as data for the logic board associated with the unit, where each may 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
*TM L	Machine type and model
*SN L	Serial number (matches external label).

## Optional Descriptors

The optional device descriptors are as follows:

Keyword	Description
*PN L	Part number
*EC L	EC level
*RL L	ROM level and ID (if ROM is present)
*LL L	Loadable ROM level and ID (minimum level required)
*MF L	Manufacturer.



---

## Micro Channel Adapter Requirements

The preferred method of implementation is to use Programmable Option Select (POS) register sub-addressing 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 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 vital product data 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 RISC System/6000 have implemented the preferred method with the required keywords defined in the following list:

- Required Words

Keyword	Description
*PN L	Part number
*EC L	EC level
*FN L	FRU number for field replacement unit
*SN L	Serial number
*DS L	Brief description, (for example SCSI, token ring, and 8-port)
*MF L	Manufacturer and location
*DD L	Device driver level
*DG L	Diagnostic level

- Optionally Required

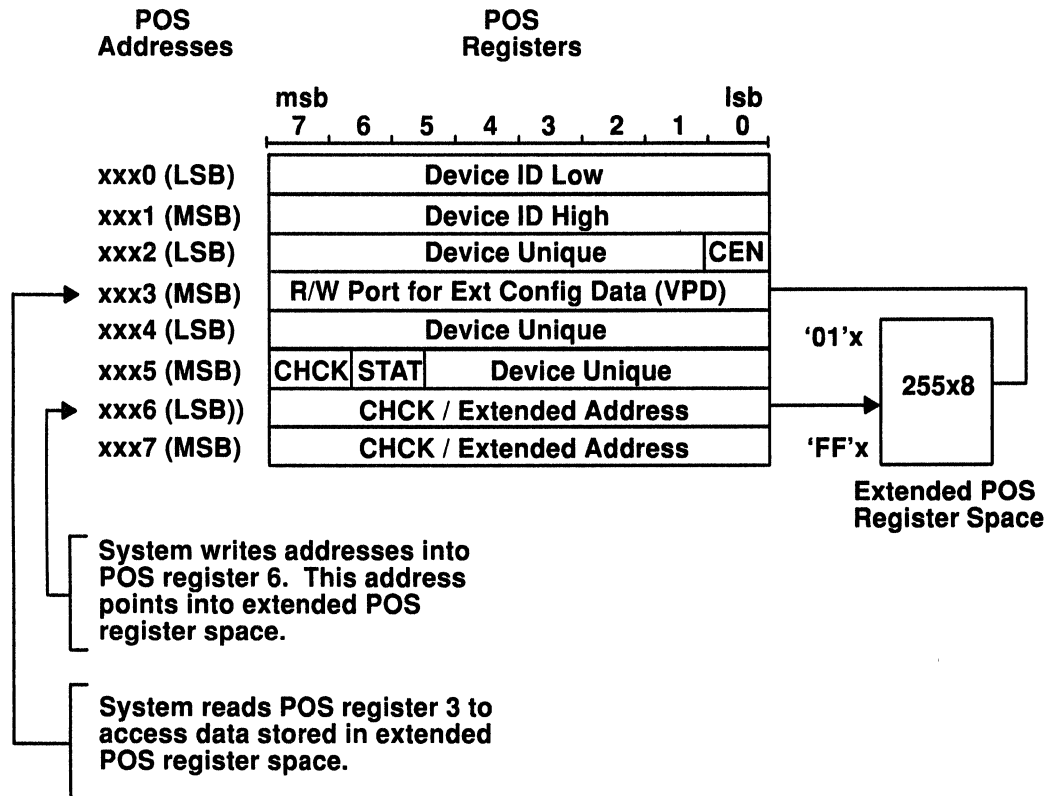
Keyword	Description
*RL L	ROM level and ID information (if ROM is present)
*LL L	Loadable microcode level (if loadable code is present)
*NA L	Network address (if adapter type requires a network address).



- Optional

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



**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 82. POS Configuration Registers

Where:

Term	Description
MSB	Most significant byte
LSB	Least significant byte.



The main objectives of the POS feature is to:

- Eliminate switches on the processor board, I/O Board and feature boards.
- Permit installation of multiple, identical-feature boards.
- Positively identify any board by slot.
- Resolve resource assignment conflicts.
- Provide access to an extension POS register that provides information specific to that feature board.

## System Configuration Protocol

Software can access the previously defined POS registers in the adapter address space. Each board decodes the three least significant bits of the address bus (X'A2', X'A1', X'A0') and the '-cd setup' signal. Normally, each enabled board returns the '-cd sfdbk' signal to the processor when an access is made to the board address space. The '-cd sfdbk' signal is not generated when the '-cd setup' line is in the setup state for addresses X'XXX0' through X'XXX7'. Boards should fully decode the X'A0', X'A1', and X'A2' addresses with the '-cd setup' signal to select a POS register.

Under this implementation, the following POS registers and bits are architected:

1. POS address X'XXX0' is the least significant byte of the adapter device ID.
2. POS address X'XXX1' is the most significant byte of the device ID in bits 0 through 3, while bits 4 through 7 specify the device function relative to the I/O bus.
3. POS address X'XXX2'. Bit 0 (the least significant bit) is the board enable bit. Other bits may be used for device unique information.

Following a reset, bit 0 is off (board disabled).

4. POS address X'XXX3' is a port that provides read access to vital product data during board setup. Optionally, port 3 may also be used for write capability beyond those bytes required for VPD up to a total of 64K bytes. POS registers 6 and 7 act as a pointer to this space.

When POS registers 6 and 7 are a value of 0, POS register 3 may contain device-unique information.

5. POS address X'XXX4' may be used for device-unique information.
6. POS address X'XXX5'. Bits 5 through 0 can be used for device-unique information. Bits 7 and 6 (most significant bits) have special uses.

Bit 7 is the channel check indicator which is set to a value of 0 on a channel check condition. The indicator is set to a value of 1 on a channel reset.

If the channel check active indicator is used by an attachment, bit 6 of POS address X'XXX5' can be used to indicate that additional status is available through POS register addresses X'XXX6' and X'XXX7'.

Bit 6 of POS address X'XXX5' is the channel check status indicator (Stat). When set to a value of 0, this bit indicates that channel check exception status is available using POS addresses X'XXX6' and X'XXX7'. When set to a value of 1, it indicates that no status is available. For status information, POS addresses X'XXX6' and X'XXX7' can be the status, a pointer to status at another address, or a command port to present the address elsewhere.



7. POS addresses X'XXX6' and X'XXX7' can be used to support exception status as previously described.

When the '-cd setup' signal is active, POS register 6 and POS register 7 are used as a pointer to an extended storage facility. POS register 6 represents the low-order byte of that address, while POS register 7 contains the high-order byte of that address pointer. If 255 or fewer bytes of data are required, POS address X'XXX7' need not be implemented. However, writing a value of 0 to POS address X'XXX7' must not affect other POS registers and normal operation.

Following a reset operation, POS addresses X'XXX6' and X'XXX7' are both reset to a value of 0.

The data located using the extended addressing mode is accessed according to the system configuration protocol for Micro Channel feature boards utilizing the 'CD Setup' line. A valid POS byte-selecting address is driven on the bus where only the least significant address bits ( X'A0' through X'A2' ) are meaningful.

Under non-exception conditions, when POS register addresses X'XXX6' and X'XXX7' are nonzero, a data byte read from POS address X'XXX3' accesses an extended storage facility within the adapter.

If POS addresses X'XXX6' and X'XXX7' are a value of 0, POS registers 3 and 4 may be accessed as device-unique Read-Write POS bits.

## Extended Storage Facility

The extended storage facility (extended configuration information) is a read-only device of sufficient size to store the information identified in Figure 83. Location 0 of this facility is not accessible since POS registers 6 and 7 must be nonzero to access this facility. Therefore, this storage facility begins at address X'00 01' (in POS registers 7 and 6, respectively), or optionally at address X'FF 01'.

POS		
7	6	
00	00	Not accessible
00	01	VPD in ASCII
00	04	TL (Length of storage in 2-byte words)
00	06	CRC (A 2-byte CRC value)
00	08	* (Delimiter)
00	09	KW (A 2-character keyword)
00	0B	L (A 1-byte inclusive length)
00	0C	Data
00	W W	* KW L (Next keyword)

Figure 83. Extended Storage Facility

The following fields are required information for every adapter supporting this architecture. A sample layout is included on page 5-17.

1. VPD: The ASCII characters "VPD" specify that this adapter supports the following architected information.

If "VPD" are not the first three characters, the data obtained is not treated as vital product data in machine-readable form.



2. 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.
3. CRC Value – This 2-byte value is a CRC value starting at address X'00 08' through the end of storage (TL).  
The CRC polynomial is  $1 + X (\text{exp } 5) + X (\text{exp } 12) + X (\text{exp } 16)$ , 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) XX YY
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. XX YY is the CRC value on data from X'00 08' through X'00 57'.
2. –, a dash, is binary zeros.
3. ( ) is 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 6. Initial Program Load (IPL) ROM

## Chapter Contents

Description .....	6-3
ROM Hardware .....	6-3
Hardware Initialization .....	6-3
Cold System Reset .....	6-3
Warm System Reset .....	6-4
ROM Warm IPL Function .....	6-4
Hardware–Initiated IPL .....	6-4
Software–Initiated IPL .....	6-4
Check Stop .....	6-4
LEDs .....	6-5
NVRAM .....	6-5
IPL Expansion Code .....	6-5
IPL Record .....	6-5
Security .....	6-5
Service IPL .....	6-5
IPL ROM Components .....	6-6
Initial Sequence Controller .....	6-6
Initial Sequence Controller Functions .....	6-7
Core Sequence Controller .....	6-7
Core Sequence Controller Functions .....	6-8
IPL Controller .....	6-8
IPL Controller Functions .....	6-10
IPL Devices .....	6-11
Trusted (Normal) Default IPL Device Selection Sequence .....	6-11
Service Default IPL Device Selection Sequence .....	6-12
Power–On Self Tests .....	6-12
POST Functional Descriptions .....	6-13
Device Interface Routines .....	6-13
Device Interface Routine Functional Description .....	6-13
IPL ROM Functional Characteristics .....	6-14
Cold IPL Entry Point .....	6-14
ROM Warm IPL Entry Point .....	6-14
IPL Control Block .....	6-14
IPL Record .....	6-15
Interface to the Loaded Code .....	6-15
NVRAM .....	6-16
IPL Expansion Code .....	6-16
Header .....	6-16
Code Area .....	6-16
Linkage Information .....	6-16
LED Operation .....	6-17
Errors .....	6-17
ROM LED Values During IPL .....	6-17
ROM Entry Point Table .....	6-17



Error Codes .....	6-18
OCS Display Codes .....	6-18
Bist Error Codes .....	6-19
IPL ROM LED Codes .....	6-20



---

## 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 that 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'.
- ROM configuration information is contained in ROM.

The following configuration information is required:

- Board engineering change (EC) level and part number
- Processor serial number
- ROM part number and ID
- ROM copyright
- ROM time stamp.

## Hardware Initialization

Prior to execution of IPL ROM code, hardware initialization puts the processor into a known working state:

- Memory Configuration registers are set to all zeros.
- Cache lines are marked as invalid entries (zero).
- Cache directories contain even parity.
- Translation Look-Aside Buffer (TLB) entries contain even parity.
- Registers contain even parity.
- Machine State register (MSR) has IP equal to 1 (Interrupt Prefix bit), ME equal to 1 (Machine Check Enable bit), and the other bits equal to 0. (The Shift Register latch (SRL) for IP equals 0 and ME equals 0).

For RISC System/6000 units with the On Card Sequencer (OCS), hardware initialization is performed by the OCS. Hardware initialization is performed before control is passed to IPL ROM code and leaves the machine in the same state for ROM.

## Cold System Reset

Cold system reset occurs at initial startup or 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 1. 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 (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.

## Software-Initiated IPL

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 RISC System/6000 units without OCS, (a check stop event causes a halt), the check stop count in NVRAM is always a value of 0.

A check event stop 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 has not occurred. The IPL ROM continues normal execution.
- A value of 1 indicates that a check stop event has occurred and that OCS has 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, that was undetected 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 RISC System/6000 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 RISC System/6000 units have at least 8K byte of NVRAM.

The IPL ROM reads the following information from NVRAM if NVRAM is valid:

- Information as to whether a memory bit should be steered and what the bit is.
- IPL expansion code
- Check stop count
- Normal device list
- Service device list.

## IPL Expansion Code

The IPL ROM code function can be expanded by way of code in NVRAM. The IPL ROM provides for the presence of an expansion code in NVRAM. The IPL sequence controller detects the presence of an expansion code, copies it in to RAM, and passes control to it.

## IPL Record

In order to IPL, a valid IPL record must reside on a valid IPL RISC System/6000 media. This record consists of:

- An ID uniquely identifying it as a RISC System/6000 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 of 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:

- Disabling of Reset button is a hardware function.
- 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 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 is loadable 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.

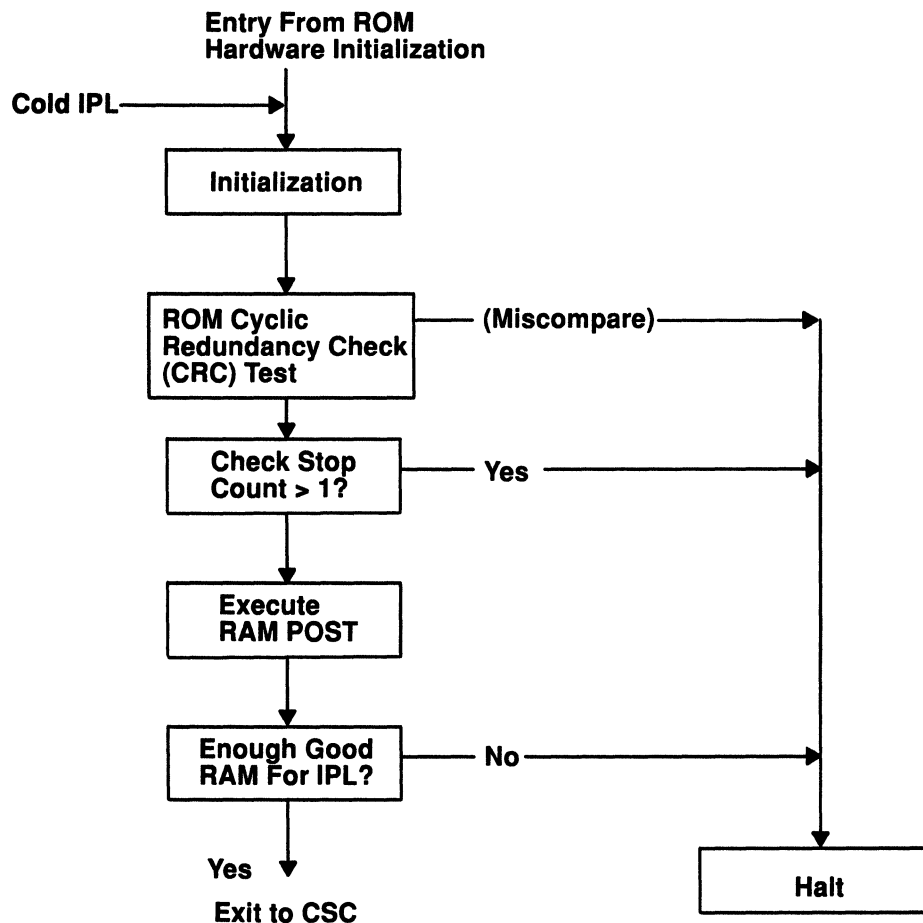


Figure 84. Initial Sequence Controller Logic Flow



## Initial Sequence Controller Functions

The major Initial Sequence controller functions are:

1. Perform initialization.
  - Read ROM configuration information from non–CRC checked part of ROM and set ROM size and speed in the Memory Control Unit Control register (MCCR).
  - Set initial LED values.
  - Perform other initialization as required.
2. Inspect check stop count:
  - If 0 or 1, continue normal execution.
  - If greater than 1, halt with an error code in LEDs.
3. Check system ROM cyclic redundancy check.
  - Halt if miscompare.
4. Execute RAM POST.
  - Determine memory configuration (includes setting CRC).
  - Find at least 1M byte of good memory (Swapping extents if necessary).
  - Test memory and create a bit map.
  - Store results of RAM POST into the IPL control block.
5. Inspect return code from the RAM POST.
  - Halt if less than 1M byte of good memory.

## 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.

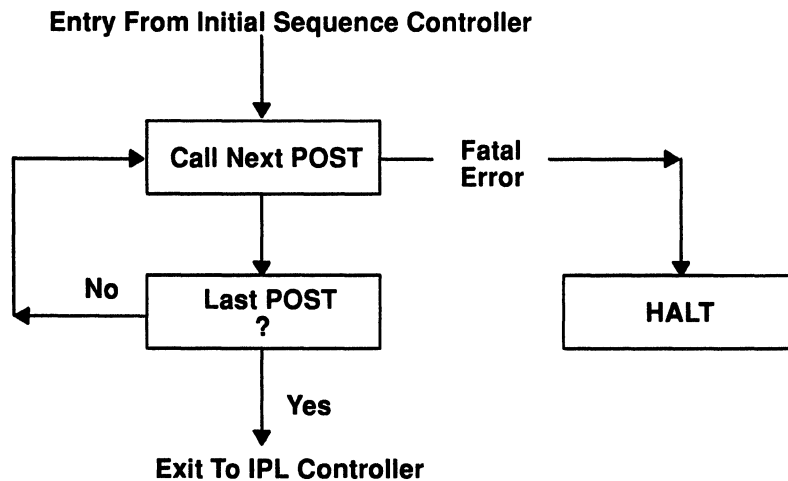


Figure 85. Core Sequence Controller



## **Core Sequence Controller Functions**

The functions of the Core Sequence controller are:

1. Execute POSTs in a predefined order.
2. POSTs are passed a pointer to the IPL control block to record results.
3. Return codes are passed 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 no IPL attempt is successful, the IPL controller continues to cycle through the IPL device list (DevList), trying to initiate an IPL from each IPL device.



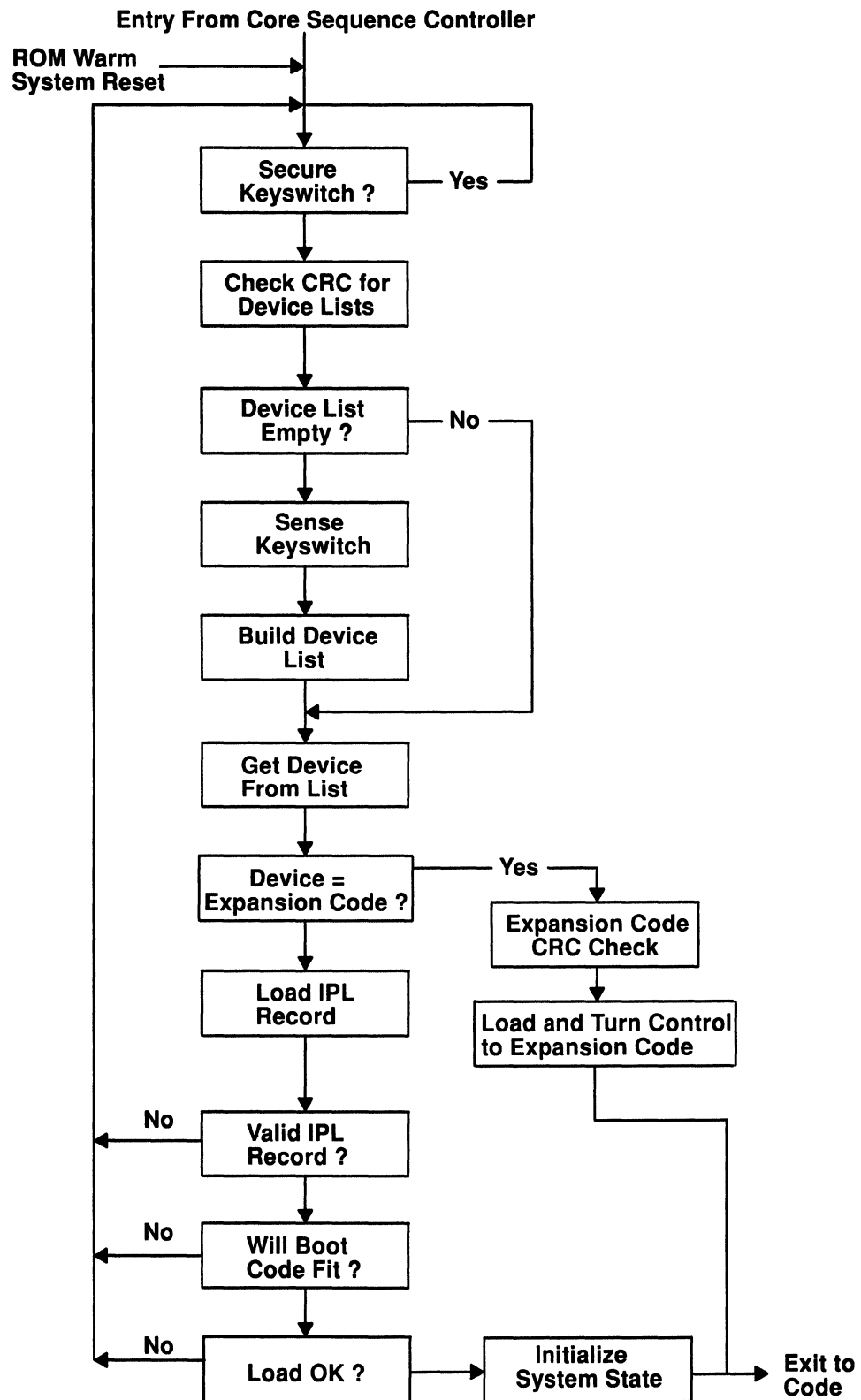


Figure 86. IPL controller



## **IPL Controller Functions**

The functions of the IPL controller are:

1. NVRAM CRC test.
  - Check NVRAM cyclic redundancy check on portions of NVRAM containing configured IPL device selection sequence.
2. Build the list of IPL device candidates based on:
  - Keyswitch position
  - Device lists (if present).
3. Cycle through created device lists.
4. Get the candidate from the list.
5. If the candidate is expansion code, load and execute it.
6. Otherwise attempt to load IPL record from candidate device. (If the Small Computer Systems Interface (SCSI) disk, the IPL controller finds a memory area to store the bad block map.)
7. If the keyswitch is not in the service position, look for an IPL record in which the primary code description length field is not 0.
8. If the keyswitch is in the service position, look for an IPL record in which the alternate code description length field is not 0.
9. If the valid IPL record is not loaded, get the next candidate from the list.
10. If all candidates have been attempted, rebuild the list and retry.
11. Load code. The code loaded in the first good 1M byte of memory is loaded contiguously. Beyond the 1M byte 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, get the next candidate from the list.
  - If all candidates have been attempted, rebuild the list and retry.
12. Initialize machine state for execution of loaded code.
13. If an IPL was done from a disk, the volume ID (unique ID) is saved in the IPL control block.

The system is left in real mode with:

- Interrupts disabled
- All good memory initialized with good error checking and correction (ECC)
- Any IPL device used inactive
- Memory contents as shown in Figure 87.



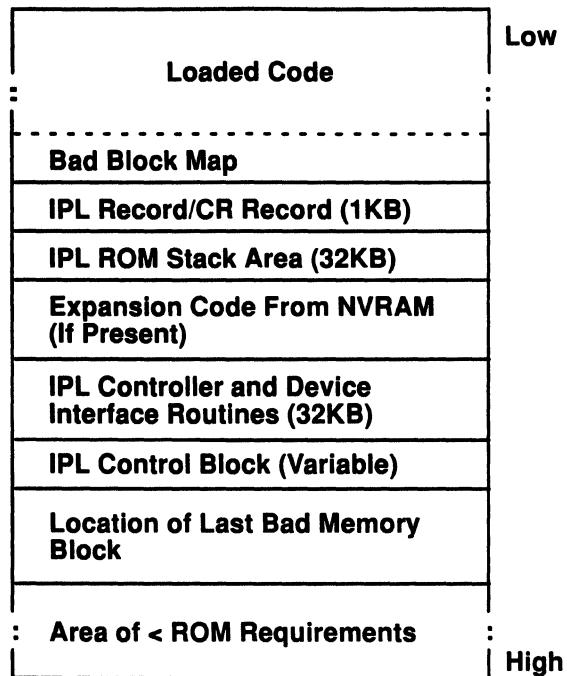


Figure 87. RAM map

14. Pass control to code loaded.

Parameters passed to the loaded code in registers are:

- 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
- IBM 7012 Model 320 direct bus–attached (DBA) file disk
- SCSI adapter–attached IPL devices
- Expansion code.

### Trusted (Normal) Default IPL Device Selection Sequence

The trusted (normal) default IPL device order is:

- Direct bus–attached file (IBM 7012 Model 320 fixed disk)
- SCSI device.



### **Service Default IPL Device Selection Sequence**

The default service IPL device list is as follows:

- Standard I/O diskette 0, and then 1
- Expansion code in NVRAM
- DBA file (IBM 7012 Model 320 fixed disk)
- SCSI device.

## **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 that portion 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 that portion of the machine necessary for 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 tests an adapter's functionality and device presence.

The following tests are performed:

- RAM POST
- I/O Channel controller (IOCC) POST
- IPL device POSTs:
  - Standard/feature diskette drive
    - Adapter test is performed.
    - Device presence test is performed.
  - IBM 7012 Model 320 DBA disk
    - Adapter ID is determined and saved.
    - Adapter test is performed.
    - Device presence test is performed.
  - SCSI disk.
    - Adapter IDs are determined and saved.
    - Adapter tests are performed.
    - Device presence tests are performed.

Before calling a POST routine, the controller puts a value in the LEDs identifying the POST so that if there is an error while running a POST and control does not return, the error is identifiable.

POST routines are passed a pointer to the area of the IPL control block in which to store the test results.



## POST Functional Descriptions

- 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)
  - DMA test
  - Test interrupts
- IPL device POSTs
  - Standard diskette POST
    - Adapter test
    - Device presence test
  - IBM 7012 Model 320 DBA disk POST
    - Adapter test
    - Device presence test
  - SCSI adapter POSTs
    - Adapter tests
    - Device presence tests.

## Device Interface Routines

IPL ROM contains device interface routines for the IPL devices. These device interface routines provide functions to enable ROM IPL code to load the IPL record and code described in the IPL and configuration records. The fixed disk device interface routines support bad block management to allow reading non-contiguous areas from the IPL media.

The device interface routines are as similar as possible for all devices. The device interface routines convert locations and lengths, as required, to the format expected by the adapters. The device geometry from the IPL record is available for use by the device interface routines used in performing the required conversions. The device interface routines should not query devices for their geometry.

## Device Interface Routine Functional Description

- Standard diskette
  - Restore parameters: IPL control block pointer, adapter identifier, device identifier.
  - Read (with retries) parameters: IPL control block pointer, cylinder, head, sector, number of sectors, memory address.
- IBM 7012 Model 320 DBA disk
  - Restore parameters: IPL control block pointer, adapter identifier, device identifier.
  - Read (with retries) parameters: IPL control block pointer, physical sector number, number of sectors, memory address.



- SCSI adapter
  - Restore parameters: IPL control block pointer, adapter identifier, device identifier.
  - Read (with retries) parameters: IPL control block pointer, physical sector number, number of sectors, memory address.

---

## 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 register 3. The pointers in the IPL control block are considered valid and reuseable.

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 RISC System/6000 linkage conventions and the register conventions established by the IPL ROM must be followed.
- The IPL ROM code alters 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 (is a unique structure for each POST)
- Results of expansion code CRC test
- A pointer to a memory bit map
- Pointer to NVRAM expansion code
- 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 a RISC System/6000 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 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 following parameter, pointer to IPL control block, in register 3.



## NVRAM

All machines have NVRAM. NVRAM is described on page 6-5.

The following are read from NVRAM by ROM IPL code:

- Check stop count (stored by hardware)
- Device lists stored by software ( trusted and service )
- IPL expansion code from software data area
- Cyclic Redundancy Check (CRC) values for the areas of NVRAM from which data is read by the IPL ROM.

For details on how the device lists are stored in the NVRAM and how the expansion code is loaded by the software, see the description of the **Nvload** and **Iplist** commands in the *Commands Reference*. For examples of device driver code refer to *Kernel Extensions and Device Support Programming Concepts* .

## IPL Expansion Code

The RISC System/6000 IPL ROM facilitates expansion of the functions performed during the execution of the IPL ROM by providing for the presence of expansion code in NVRAM. When present, the expansion code is loaded in RAM and executed in-line as part of the normal IPL ROM operation.

The expansion code resides in the variable software area of NVRAM. IPL expansion code contains a header and a code area. The IPL ROM uses the header to detect the presence and validity of the expansion code. The code area includes the code to be run as an extension to the in-line IPL ROM operation. It has complete control of the machine when run.

### Header

The header is a field that contains the recognition pattern, the length of the code area, the pointer to the expansion code in NVRAM, and the CRC check value for the code area. The recognition pattern consists of two bytes containing X'A5A5'. The CRC is a field containing the CRC value for the code area. The length field is a 2- byte field containing the size of the code area in bytes.

### Code Area

The first word of the code area must be the entry point. If an expansion code header is present in NVRAM, the code area must be present even if it merely does a return to the normal IPL ROM.

### Linkage Information

After the expansion code area is recognized, the code is loaded into the RAM location and the CRC is verified. Then ROM passes control to the entry point. The expansion code has complete control of the system at this point.

The expansion code is expected to complete the IPL sequence without returning to ROM.

The expansion code must adhere to the common linkage conventions described in *AIX Version 3 Technical Reference Manual*.

Arguments are passed to the expansion code in the general purpose register R3 which is a pointer to the IPL control block.

The ROM cannot handle a return from any IPL device.



## 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:

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 will not be visible to the operator.
3. If the POST code gets lost, 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 non-fatal error.
5. If the error is non-fatal, 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 non-fatal. The fatal errors are those that prevent an IPL. Non-fatal 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 is needed to be presented in the LEDs.

The LED codes displayed during execution of the IPL ROM are listed on page 6-20.

## 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

The error codes displayed by the LEDs are listed in the following three categories:

- OCS display codes
- BIST error codes
- IPL ROM LED codes.

### OCS Display Codes

The following is a list of the OCS display codes:

Code	Description
100	Success code.
101	Starting initialization of BIST (built in self test).
102	Starting BIST.
103	Model not found.
104	Equipment conflict: CBA not found.
105	Cannot read OCS EPROM.
112	Checkstop but cannot log out.
113	Checkstop count > 0.
120	Starting CRC check.
121	Bad CRC on OCS EPROM.
123	Bad CRC on OCS NVRAM space.
125	Bad CRC on TOD RAM.
140	Bad configuration of manufacture's bits.
142	Box manufacturing.
144	Manufacturing BIST error.
151	AIPGM test code.
152	DCLST test code.
153	ACLST test code.
154	AST test code.
180	LOGOUT test number.

Code	Chip
001	FPU.
002	FXPT.
003	ICU.
004	MCU.
010	DCU0.



<b>029</b>	IOCC01.
<b>042</b>	DCU1.
<b>061</b>	IOCC02.
<b>074</b>	DCU2.
<b>106</b>	DCU3.

## **BIST Error Codes**

The FAILCDS field records module errors. The following summarizes the possible error codes that might be recorded here:

<b>Code</b>	<b>Description</b>
<b>0</b>	No errors at all (a good module).
<b>1</b>	DCLST has error on this module.
<b>2</b>	ACLST has error on this module.
<b>4</b>	AST has error on this module.
<b>8</b>	The DD level could not be ascertained for this module.
<b>16</b>	The module is not present or is not responding on the COP bus.
<b>32</b>	An equipment incompatibility exists between this module and another.

Any combination of these error codes is possible, and is displayed in the decimal radix by the OCS as a BIST error. For example, suppose a module passed DCLST and ACLST, but has an error in AST, then the error code displayed would be the three-digit CBA of the module, followed by the two-digit error code, 04. If the module has an error in both ACLST and AST, the error code would be 06.



## IPL ROM LED Codes

The following is a list of the IPL ROM LED codes.

**Note:** The following LEDs are normally displayed for 1 second or less, unless noted.

LED	Fate	Description
200		Keylock in secure position.
201	Fatal	IPL ROM initialization.
202	Fatal	Machine check handler.
203	Fatal	Data Storage Interrupt handler.
204	Fatal	Instruction Storage Interrupt handler.
205	Fatal	External Interrupt handler.
206	Fatal	Alignment Interrupt handler.
207	Fatal	Program Interrupt handler.
208	Fatal	Floating point unavailable handler.
209	Fatal	Reserved 900 handler.
210	Fatal	SVC 1000 handler.
211	Fatal	IPL ROM CRC did not compare correctly.
212	Fatal	RAM POST Memory Configuration registers error.
213	Fatal	RAM POST full or halfword and byte load and store error.
214	Fatal	RAM POST PIO load and store circuitry error.
215	Fatal	RAM POST ECC generation circuitry error.
216	Fatal	RAM POST ECC correction circuitry error.
217	Fatal	RAM POST bit steering logic error.
218	Fatal	RAM POST 1 M byte of good memory not found, address or remap error.
219	Fatal	RAM POST bit map generation.
220	Fatal	IPL control block initialization.

**Note:** The following LEDs may be displayed for 10 seconds or more.

LED	Keyswitch Position	IPL Path Set By	Attempting Device
221	Normal	NVRAM	NVRAM CRC checked bad.
222	Normal	NVRAM	Standard I/O selected for IPL.
223	Normal	NVRAM	SCSI devices selected for IPL.
224	Normal	NVRAM	Synchronous Link Adapter selected for IPL.
225	Normal	NVRAM	IBM 7012 Model 320 (DBA fixed disk) selected for IPL.
226	Normal	NVRAM	Ethernet selected for IPL.
227	Normal	NVRAM	Token ring selected for IPL.
228	Normal	NVRAM	Expansion code selected for IPL.
232	Normal	ROM	Reserved for Standard I/O usage.
233	Normal	ROM	SCSI devices selected for IPL.
234	Normal	ROM	Synchronous Link Adapter selected for IPL.
235	Normal	ROM	IBM 7012 Model 320 (DBA fixed disk) selected for IPL.
236	Normal	ROM	Ethernet selected for IPL.
237	Normal	ROM	Token ring selected for IPL.
241	Service	NVRAM	Reserved for service mode.
242	Service	NVRAM	Standard I/O selected for IPL.
243	Service	NVRAM	SCSI devices selected for IPL.
244	Normal	NVRAM	Synchronous Link Adapter selected for IPL.



<b>245</b>	<b>Service</b>	<b>NVRAM</b>	<b>IBM 7012 Model 320 (DBA fixed disk) selected for IPL.</b>
<b>246</b>	<b>Service</b>	<b>NVRAM</b>	<b>Ethernet selected for IPL.</b>
<b>247</b>	<b>Service</b>	<b>NVRAM</b>	<b>Token ring selected for IPL.</b>
<b>248</b>	<b>Service</b>	<b>NVRAM</b>	<b>Expansion code selected for IPL.</b>
<b>252</b>	<b>Service</b>	<b>ROM</b>	<b>Standard I/O selected for IPL.</b>
<b>253</b>	<b>Service</b>	<b>ROM</b>	<b>SCSI devices selected for IPL.</b>
<b>254</b>	<b>Service</b>	<b>ROM</b>	<b>Synchronous Link Adapter selected for IPL.</b>
<b>255</b>	<b>Service</b>	<b>ROM</b>	<b>IBM 7012 Model 320 (DBA fixed disk) selected for IPL.</b>
<b>256</b>	<b>Service</b>	<b>ROM</b>	<b>Ethernet selected for IPL.</b>
<b>257</b>	<b>Service</b>	<b>ROM</b>	<b>Token ring selected for IPL.</b>

**Note:** The following LEDs are normally displayed for 1 second or less, unless noted.

<b>LED</b>	<b>Fate</b>	<b>Meaning</b>
<b>290</b>	<b>Fatal</b>	<b>IOCC POST</b>
<b>291</b>		<b>Standard I/O POST</b>
<b>292</b>		<b>SCSI POST</b>
<b>293</b>		<b>IBM 7012 Model 320 POST</b>
<b>294</b>		<b>Synchronous Link Adapter POST</b>
<b>295</b>		<b>XX3 POST (Ethernet)</b>
<b>296</b>		<b>XX2 POST (Token ring)</b>
<b>297</b>		<b>XX1 POST</b>
<b>298</b>		<b>Attempting warm IPL</b>
<b>299</b>		<b>IPL ROM has completed loading and has passed control to loaded code (loaded code changes LEDs).</b>







---

# Chapter 7. Keyboard/Tablet/Speaker Adapter

## Chapter Contents

Description .....	7-5
System Interface: Input/Output Operations to Adapter .....	7-7
Sequences of Events for System/Adapter Communications .....	7-7
System Initiated Transfer to the 8051 Adapter .....	7-7
Adapter's 8051 Initiated Transfer to System .....	7-8
Adapter's 8051 Initiated Block Transfer to System .....	7-8
IOW and IOR Operations .....	7-9
Read 8255 PA Input Buffer .....	7-9
Read Command Register .....	7-9
Read 8255 PB Port .....	7-10
Read 8255 PC Register .....	7-10
Write 8255 PA Output Buffer .....	7-11
Configure 8255 Chip .....	7-11
Enable Keyboard and UART IRQ .....	7-12
Disable Keyboard and UART IRQ .....	7-12
Adapter Reset Operation .....	7-12
Activate Adapter Reset .....	7-12
Release Adapter Reset .....	7-12
Adapter Initiated Interrupt Request – ID Codes .....	7-13
Adapter Commands .....	7-14
Command Byte Decodes .....	7-14
Extended Command Decodes .....	7-15
Select Extended Command Set (X'00') .....	7-15
Write to Keyboard (X'01') .....	7-16
Write to Speaker (X'02') .....	7-16
Write to UART Device (X'03' and X'04') .....	7-16
Write UART – Control (X'03') .....	7-17
Write UART – Query (X'04') .....	7-17
Set UART Baud Rate (X'05') .....	7-17
Initialize UART Framing (X'06') .....	7-18
Set Speaker Duration – High Byte (X'07') .....	7-18
Set Speaker Frequency – High and Low Byte (X'08' and X'09') .....	7-18
Diagnostic Write Keyboard Port Pins (X'0C') .....	7-19
Write Shared RAM (X'1R') .....	7-19
Extended Command Descriptions .....	7-19
Read Shared RAM (X'00' – X'1F') .....	7-19
Reset Mode Bit (X'20' – X'2F') .....	7-19
Set Mode Bit (X'30' – X'3F') .....	7-19
Initialize Speaker Volume (X'40' – X'43') .....	7-20
Terminate Speaker and Reset Duration (X'44') .....	7-20
Set Scan Count for System Attention Keystroke Sequence (X'5S') .....	7-20
Execute 8051 Soft Reset (X'60') .....	7-20
Force System Attention Interrupt (X'62') .....	7-21
Diagnostic Sense Keyboard and UART Port Pins (X'70') .....	7-21



Dump Adapter Shared RAM (X'80' and X'81')	7-21
Dump RAS Logs With or Without Reset (X'82' and X'83')	7-22
Restore Initial Conditions (X'90')	7-22
Read 8051 Release Marker (X'E0' – X'EF')	7-22
NOP (X'F0' – X'FF')	7-22
Functional Description and Allocation Map	7-22
Read-Shared RAM	7-23
Modes and Status Bits in Shared RAM	7-24
Read-Only Shared RAM	7-25
RAS Logs in Shared RAM	7-26
Adapter Speaker Control	7-27
Sharing of Speaker Input With the Micro Channel Audio Signal	7-27
Speaker Frequency Control	7-27
Speaker Duration Control	7-28
Speaker Volume Control	7-28
Speaker Command Queue Description	7-29
Functional Operation	7-29
Implementation	7-29
Keystroke Click Description	7-29
Functional Operation	7-30
Implementation	7-30
Click Suppression for Defined Scan Code Set	7-30
Click Interference with other Speaker Operations	7-30
Adapter RAS and Security Functions	7-31
Detection of Special Keystroke Sequences	7-31
Initiate System Attention Interrupt	7-31
Diagnostic Wraps	7-32
Adapter Self-Test After a Power-On, System, or Adapter Reset Operation	7-32
Diagnose Functions Executed on System Command	7-32
Adapter Informational Codes Returned to System	7-32
Acknowledgement Informational Codes	7-33
Command Reject Informational Codes	7-33
Status Report Informational Codes	7-34
Adapter Error Codes Returned to System	7-34
Abnormal End Codes	7-34
System Action Required	7-34
Device Error Codes	7-35
System Action Required	7-35
Codes	7-35
Keyboard Device Support Notes	7-37
Keyboard Commands	7-37
Resend (X'FE')	7-37
Echo (X'EE')	7-37
Keyboard Outputs	7-37
Resend (X'FE')	7-37
Ack (X'FA')	7-37
Overrun (X'00')	7-37
Adapter Design Notes	7-37
Channel I/O Device Address Bit Decoding	7-40
8051 RAM Allocation	7-40



Adapter and Keyboard Initialization Procedure .....	7-41
Standard I/O Adapter Board to Device Interface .....	7-43
Keyboard Port Interface .....	7-43
Tablet (UART Port) Device Interface .....	7-43







---

## Description

The adapter contains an 8051 single-chip microcontroller programmed to support a RISC System/6000 family serial keyboard interface, a tablet device with a full duplex serial UART interface, and a speaker. The adapter interfaces bi-directionally to the system bus through an 8-bit 82C55A Programmable Peripheral Interface chip, and a 6-bit Command register for keyboard and UART ports.

The adapter also performs a multi-keystroke detection, which directly initiates a system attention interrupt.

The Standard I/O adapter board provides the connectors to the keyboard, tablet devices, and speaker. These functions are not available on some RISC System/6000 system units.

## System Software Interface

System-to-adapter communications are over the two low-order data bus bytes (D0 through D15) utilizing I/O write operations. Adapter-to-system communications are initiated by the adapter raising an interrupt level request. Data transfers to the system are over the byte (D0 through D7) utilizing I/O read operations. Additionally, an adapter selective reset can be initiated by the system using an I/O write sequence.

## Programmable Peripheral Interface Functions

The 82C55A (referred to hereafter as 8255) chip is a general-purpose programmable I/O chip used to interface the 8051 microcontroller to the system, and as a diagnostic sensing port. The 8255 chip is configured to operate in mode 2 with ports B and C (lower) defined as inputs. Port A provides a bi-directional bus with the 8051 chip while port C (upper) provides the necessary handshake controls.

## Microcontroller Functions

The adapter generally does not interpret information passing between the system and the keyboard or tablet. That information is simply passed through the adapter. The 8051 receives data from the keyboard or tablet, validates it, and passes it on to the system. The 8051 chip receives device commands from the system and passes them on to the selected device. The 8051 chip also receives commands from the system that are validated and executed by the 8051 chip, for example, reading and writing shared random access memory (RAM).

The 8051 chip has 128 bytes of RAM. Commands are defined that allow the system to read or write a selected set of these bytes. This shared RAM contains status and mode control information, error logs, and device control parameters for the keyboard, speaker and UART port.

## Keyboard Interface

The adapter receives a frame of data serial by bit from the keyboard, validates the frame, buffers up to 5 bytes, and presents the data to the system as a byte of data in the 8255 input buffer. The adapter interrupts the system when data is placed in the interface buffer.

The system can transmit keyboard commands to the keyboard by writing to the Command register of the adapter and 8255 output buffer. The 8051 chip embeds the data byte in a frame, and then transmits the frame serial by bit to the keyboard with odd parity.

The 11-bit framing protocol and keyboard commands are defined in chapter 8 of this manual. Scan codes received from the keyboard are passed on to the system in their original form; the adapter performs no translation. However, keystrokes received while the buffer is full (overrun) are lost.



## **Tablet Interface**

The adapter provides a port for tablet devices. It is a full duplex serial port operating in an 11-bit UART framing protocol. The adapter buffers up to two blocks (or reports) of tablet device data through the UART port. The system can send commands through the adapter to the tablet. Data reports and commands are defined in the tablet section of this manual.

Reports received from the tablet are passed on to the system in their original form; the adapter does not change the report byte content. However, reports received while the buffer is full (overrun) are lost.

## **Speaker Interface**

The system keyboard contains a small speaker that can be driven from either the adapter or the Micro Channel audio signal. The adapter always has control of the volume. Frequency and duration parameters are controlled by either the adapter or the Micro Channel audio signal through an 'OR' circuit. System software must ensure that only one path is active at a time.

The adapter provides an option of automatic click tone through the speaker for every keystroke at a default frequency and duration.

The adapter buffers one set of frequency and duration parameters while a current tone is active.



---

## System Interface: Input/Output Operations to Adapter

System software communicates with adapter by using memory-mapped read and write operations. Configuring the 8255 chip, enabling, and disabling interrupt requests are done with a write operation to the 8255 chip. The adapter hardware reset function is performed by writing to the Standard I/O Programmable Option Select (POS) registers.

Communications between the system software and the adapter can be initiated by either party. The system can send 2 bytes of command and data to the adapter. The adapter has two modes of transferring data to the system: non-blocked (single byte) and blocked (multiple bytes in sequence). The adapter signals the system that it has information to transfer with an interrupt request, if enabled. Alternatively, the system can disable the interrupt request and poll the 8255 chip for keyboard and UART port data.

### Sequences of Events for System/Adapter Communications

The following paragraphs list the specific, detailed steps used for communications between the system software and the 8051 adapter.

#### System Initiated Transfer to the 8051 Adapter

1. System verifies that the 8255 output buffer is not full (by testing its internal flag).
2. System should set an internal programming flag indicating that the 8255 output buffer is full. (The system should reset the flag in an acknowledgement interrupt handler).
3. System issues an I/O write operation to the adapter with the command and data bytes.
4. The 8255 chip sets the output buffer full (OBF) [PC7 = 0], which initiates an 8051 interrupt.
5. The 8051 interrupt reads the Command register (command must be read before data).
6. The 8051 interrupt reads the data from the 8255 PA register.
7. The 8255 chip resets the OBF [PC7 = 1], dropping the interrupt to the 8051 chip.
8. The 8051 interrupt posts the command and data to a work queue.
9. The 8051 chip exits the interrupt level.
10. The 8051 command execution work queue initiates an interrupt request to the system signifying execution status of the command:
  - ID 0: Informational (command accepted or rejected)
  - ID 3: Returning requested byte
  - ID 4: Requested block transfer ready.



### **Adapter's 8051 Initiated Transfer to System**

1. The 8051 chip waits for the input buffer to empty (IBF = 0) by testing PC5.
2. The 8051 chip writes interrupt ID to 8255 PC bits 2–0. (PC must be written before PA.)
3. The 8051 chip writes a data byte to the 8255 PA input buffer.
4. The 8255 chip sets the IBF to full (IBF = 1), which raises IRQ to the system.
5. System accepts the interrupt request.
6. System reads interrupt ID from 8255 PC register. (PC must be read before PA.) If ID equals B'100'; then go to the block transfer chart.
7. System reads a data byte from the 8255 PA register (input buffer).
8. The 8255 chip resets IBF, which drops IRQ.
9. System processes information and exits the interrupt level.

### **Adapter's 8051 Initiated Block Transfer to System**

This sequence transfers multiple bytes of the following kinds of information:

- Tablet data when in blocking mode
- Adapter shared RAM dump
- RAS logs dump.

The following steps initiate an adapter 8051 block transfer to the system:

1. The 8051 chip waits for the IBF to equal 0.
2. The 8051 chip writes the interrupt ID B'100' to PC bits 2–0.
3. The 8051 chip writes a block length (byte count) to the 8255 PA input buffer.
4. The 8255 chip sets the IBF equal to 1, which raises IRQ to the system.
5. The 8051 chip waits for the IBF to equal 0.
6. System accepts an interrupt request.
7. System reads interrupt ID B'100' from the 8255 PC register.
8. System reads the count from the 8255 PA register.
9. The 8255 chip resets the IBF, which drops IRQ.
10. System waits (polls) for IRQ to be set (either by sensing IRQ internally in the system processor, or by reading the 8255 PC bit 5).
11. The 8051 chip writes an interrupt ID appropriate to the data to the 8255 PC bits 2–0 (B'010' or B'011').
12. The 8051 chip writes a data byte to the 8255 PA input buffer.
13. The 8255 chip sets the IBF equal to 1, which sets PC5 and raises IRQ to the system (if enabled).
14. The 8051 chip waits for the IBF to equal 0.
15. System reads interrupt ID from the 8255 PC register.
16. System reads a data byte from the 8255 PA register.
17. The 8255 chip resets the IBF, which resets PC5, which drops IRQ.



18.Repeat the previous 8 steps for the number of data bytes indicated by the count.

19.System processes information and exits the interrupt level.

## IOW and IOR Operations

The following chart lists the defined I/O Operations to the adapter:

Host Operation	I/O Address	Function	Comments
IOR	0054	Read 8255 PA	Input buffer returned
IOR	0059	Read command	Last command returned
IOR	0055	Read 8255 PB	Diagnostic sense
IOR	0056	Read 8255 PC	Low 3 bits = interrupt ID
IOW	0050	Write 8255 PA	Command and data latched for execution by 8051 chip
IOW	0057	Configure 8255	Required data = X'C3'
IOW	0057	Enable IRQ	Required data = X'09'
IOW	0057	Disable IRQ	Required data = X'08'
IOW	0058	Clear 3–Key system interrupt	Write any data
IOR	0058	3–Key system interrupt Status	Bit 0 = 1

### Read 8255 PA Input Buffer

Operation	Address	High–Data Byte	Low–Data Byte
IOR	0054		PA input buffer

**Action** The 8–bit 8255 PA input buffer contents are returned to system software. 8255 PC bit 3 is reset, which drops IRQ to the system processor and frees the 8255 PA input buffer.

**Pre–Condition** The 8255 PA input buffer is valid only when the 8255 PC bit 3 equals 1, which initiates an interrupt request to the system processor.

**Comments** The meaning of the returned byte is determined by the interrupt ID in PC bits 2–0, which must have previously been read.

### Read Command Register

Operation	Address	High–Data Byte	Low–Data Byte
IOR	0059		Command register

**Action** The 8–bit Command register contents are returned to system software. This register contains bits 13–8 of the last write operation to the 8255 PA output buffer.

**Pre–Condition** This register is only valid after a write operation to the 8255 PA output buffer.

**Comments** The returned byte bits 7–6 are 1s followed by bits 13–8 of the command written in bits 5–0.



## Read 8255 PB Port

Operation	Address	High–Data Byte	Low–Data Byte
IOR	0055		Port B inputs

**Action** The following eight signals wired into the 8255 Port B are sensed and returned to system software:

Bit	Description
7	Reserved
6	+Speaker volume bit 1
5	+Speaker volume bit 0
4	–ACK (Acknowledge)
3	–STB (Strobe)
2	–Tablet/Mouse fuse good
1	–Keyboard fuse good
0	UART RXD

**Pre–Condition** (None)

**Comments** This operation is used during polling operations after the IBF has been sensed in the PC register and prior to reading the PA input buffer. For diagnostic purposes, it can be issued at any time to dynamically sense the state of the eight signals listed previously.

## Read 8255 PC Register

Operation	Address	High–Data Byte	Low–Data Byte
IOR	0056		PC register contents

**Action** The 8–bit 8255 PC register contents are returned to system software. PC bit 7 indicates whether the PA output buffer is full (0) or empty (1). PC bit 5 indicates whether PA input buffer is full (1) or empty (0). PC bits 6 and 4 are used for hand shaking controls between the 8255 chip and the 8051 chip and can be ignored. PC bit 3 is the interrupt request line to the system processor. PC bits 2–0 define eight possible interrupt identifier codes for the PA input buffer as follows:

PC 7–0 (Binary)	Interrupt ID Number
XX1X i000	0
XX1X i001	1
XX1X i010	2
XX1X i011	3
XX1X i100	4
XX1X i101	5
XX1X i110	6
XX1X i111	7



Where *X* = don't care bits  
*i* = 1 if IRQ enabled, or 0 if disabled.

Refer to "Adapter Initiated Interrupt Request – ID Codes" on page 7-13 for an explanation of the interrupt IDs.

**Pre-Condition** PC register bits 2–0 are valid only if PC bit 5 equals 1. The 8255 chip resets PC bit 5 when the PA input buffer is read.

**Comments** This operation determines why interrupt request was initiated by the adapter and how the PA input buffer should be interpreted. This operation must be issued prior to reading the PA input buffer. This operation can also be used prior to writing to the PA output buffer to determine if it is full or empty.

## Write 8255 PA Output Buffer

Operation	Address	High-Data Byte	Low-Data Byte
IOW	0050	8051 command	8051 data

**Action** Two bytes of data are written to the adapter as follows (bit 15 is the most significant bit):

Bits 15–14 – Should be set to B'00'

Bits 13–8 – Latched in adapter Command register

Bits 7–0 – Latched in 8255 PA output buffer.

The 8255 chip resets PC bit 7 (to 0) indicating that the PA output buffer is full, which initiates an interrupt request to the 8051 chip. When enabled, the 8051 interrupt handler first reads the 5 low bits from the Command register (bits 12–8), then the 8 bits from the 8255 PA output buffer (bits 7–0). Then the 8255 chip sets PC bit 7 (to 1) indicating that the PA output buffer is now empty. The 13-bit command and data is posted to an 8051 internal queue for subsequent execution.

**Pre-Condition** The 8255 PA output buffer must be empty. This can be determined by interrogating the state of the 8255 PC bit 7: (0) means full and (1) means empty.

**Comments** The 14-bit Command and Data combinations are described in "Adapter Commands" on page 7-14. The subsequent 8051 execution handler acknowledges the command by initiating an appropriate interrupt ID.

## Configure 8255 Chip

Operation	Address	High-Data Byte	Low-Data Byte
IOW	0057		X'C3'

**Action** The 8255 chip resets its internal registers (Control, PA, PB, PC). Then the data byte is loaded into the 8255 Control register, setting the 8255 chip to operate as follows:

- The PA is a bi-directional port with an output buffer and an input buffer.
- The PB is an input-only sensing port for diagnostic use.
- The PC bits 7–4 are handshaking controls for the PA register to the 8051 chip.
- The PC bit 5 means the 8255 input buffer is full.



- The PC bit 3 is an interrupt request to the system processor.
- The PC bits 2–0 are input–only sensing pins set by the 8051 chip.

**Pre–Condition** The 8051 chip must be held in its reset state while this configure operation is being done.

**Comments** This operation is normally issued after the adapter has been reset, either by a reset adapter operation, a system reset, or a power–on operation.

## Enable Keyboard and UART IRQ

Operation	Address	High–Data Byte	Low–Data Byte
IOW	0057		X'09'

**Action** The 8255 PC bit 3 is allowed to initiate an interrupt request to the system processor when the PA input buffer has been loaded by the 8051 chip.

**Pre–Condition** The 8255 chip must have been properly configured.

**Comments** This operation is normally issued after the 8255 chip has been configured and after an interrupt request had previously been disabled.

## Disable Keyboard and UART IRQ

Operation	Address	High–Data Byte	Low–Data Byte
IOW	0057		X'08'

**Action** The 8255 PC bit 3 is disallowed from initiating an interrupt request to the system processor.

**Pre–Condition** (None)

**Comments** This operation can be issued to suspend system software and adapter communications; adapter and device operations are not directly affected. Overrun conditions can occur if the suspension lasts longer than the buffering capability of the devices attached, for example, the keyboard and tablet.

## Adapter Reset Operation

Resetting the adapter is a two–step process: first the reset state must be activated, then released. The minimum time that the reset must be active is 10 microseconds. There is no maximum time.

### Activate Adapter Reset

**Action** The selective reset to the adapter is raised and held active. The 8051 chip is held in its reset state. (The 8255 chip is only reset by a power–on operation.)

**Pre–Condition** (None)

**Comments** The adapter reset condition is held active until a subsequent release adapter reset operation is performed.

### Release Adapter Reset

**Action** The selective reset to the adapter is dropped. The 8051 chip proceeds to perform its internal self–testing and initialization. At the completion of the 8051 self–testing, a completion code is posted to the shared RAM address X'1C' and to the system in the 8255 chip with an interrupt ID of 6.



**Pre-Condition** The adapter selective reset was activated either by an activate adapter reset operation, by a system reset, or by a power-on reset. The 8255 chip must be configured before the reset operation is released.

**Comments** The 8051 chip performs its self-test and initializes RAM to the defined defaults. Refer to "Adapter and Keyboard Initialization Procedure" on page 7-41 for the recommended procedure to initialize the adapter and keyboard.

## Adapter Initiated Interrupt Request – ID Codes

When the 8051 adapter has information to pass to the system, a 3-bit identification code is placed in the 8255 PC register bits 2–0 and a data byte in the PA Input register. Then an interrupt request is raised to the system, if enabled. The following table lists the interrupt ID codes, and their meanings.

Interrupt ID Number	Interrupt Meaning	Data Byte Contents
0	Informational interrupt	Information code *
1	Byte received from keyboard	Received byte
2	Byte received from UART device	Received byte
3	Returning byte requested by system	Requested byte
4	Block transfer ready	Byte count
5	[Unassigned]	
6	8051 self-test performed	Completion code
7	8051 detected an error condition	Error code **

\* Refer to "Adapter Informational Codes Returned to System" on page 7-32.

\*\* Refer to "Adapter Error Codes Returned to System" on page 7-34.



---

## Adapter Commands

Commands interpreted by the 8051 chip are initiated by the system by way of an I/O write operation to the 8255 PA register (output buffer). Command byte bits 4–0 are latched in the Command register of the adapter and are decoded by the 8051 chip to determine the meaning of the data byte latched in the 8255 PA register. Command byte bits 7–5 are diagnostic controls with bit 5 latched in an extension to the Command register. The Diagnostic Control bits must normally be B'000'.

### Command Byte Decodes

The following table lists the command byte decodes.

Command	Byte	Command Function	Data Byte Function
0000	0000	Select extended command set	Extended command (see next chart)
0000	0001	Write to keyboard	Byte to be transmitted to keyboard
0000	0010	Write to speaker	Tone duration low-byte (number of 1/128 second ticks)
0000	0011	Write UART – control (no response)	Byte to be transmitted to UART device
0000	0100	Write UART – query (response expected)	Byte to be transmitted to UART device
0000	0101	Set UART baud rate	Baud rate counter value (default is 9600 bps)
0000	0110	Initialize UART framing	Odd and even parity control and blocking factor (default = X'84')
0000	0111	Set speaker duration	Tone duration high-byte
0000	1000	Set speaker frequency-high	Frequency counter high-byte
0000	1001	Set speaker frequency-low	Frequency counter low-byte
0000	1010	[Unassigned]	
0000	1011	[Unassigned]	
0000	1100	Diagnostic write keyboard port pins	Bit 6 to keyboard data out pin Bit 7 to keyboard clock out pin
0000	1101	[Unassigned]	
0000	1110	[Unassigned]	
0000	1111	[Unassigned]	
0001	RRRR	Write shared RAM	Byte written to 8051 shared RAM (low order 4 command bits select shared RAM address)



## Extended Command Decodes

The extended command set is decoded by the 8051 chip from the data byte latched in the 8255 PA register when the Command register equals X'00'. The following table lists the data byte decodes for the extended command set.

Data Byte	Extended Command Function
00–1F	Read shared RAM (low–order 5 bits equals shared RAM address)
2M	Reset mode bit number M (M = 0–F hex)
3M	Set mode bit number M (M = 0–F hex)
40–43	Initialize speaker volume: 40 = off, 41 = low, 42 = medium, 43 = high
44	Terminate speaker and reset duration
45–4F	[Unassigned]
50	[Not valid]
51–53	Set scan count for system attention keystroke sequence 51: 1 keystroke 52: 2 keystrokes 53: 3 keystrokes (default)
54–5F	[Not valid]
60	Execute 8051 soft reset – force abnormal end code X'A0'
62	Force system attention interrupt
61, 63–6F	[Unassigned diagnostics]
70	Diagnostic sense keyboard and UART port pins returned byte: bit 0 = UART RXD pin bit 2 = keyboard clock in pin bit 5 = keyboard data in pin
71–7F	[Unassigned diagnostics]
80	Dump adapter shared RAM address 00–0F
81	Dump adapter shared RAM address 10–1F
82	Dump RAS Logs, reset activity and error counters
83	Dump RAS Logs without counter reset
84–8F	[Unassigned]
90	Restore initial conditions
91–DF	[Unassigned]
E0–EF	Read 8051 release marker (low–order 4 bits = byte offset into 16–byte release marker)
FX	NOP – adapter only returns an ACK interrupt to system (X = don't care)

## Select Extended Command Set (X'00')

The adapter interprets the data byte as an extended command, which does not require an associated parameter. Refer to the specific extended command description.



## Write to Keyboard (X'01')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'41'	Command rejected; adapter is busy with a previous transmission to the keyboard (S3 = 1)
X'43'	Command rejected; keyboard interface is disabled (M11 = 0)
X'44'	Command rejected; associated data byte is not valid.

If the command is accepted, the associated data byte is transmitted to the keyboard.

The associated data byte to be transmitted to the keyboard must not be the keyboard **resend** command as defined by shared RAM address X'03'.

## Write to Speaker (X'02')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an Informational Code indicating acceptance or rejection of the command:

Code	Description
X'01'	Command accepted; speaker started
X'04'	Command accepted; parameters queued
X'47'	Command rejected; duration that is not valid specified
X'48'	Command rejected: frequency count value that is not valid specified
X'4A'	Command rejected; speaker queue full (S7 = 1).

If the speaker queue is full, the command is rejected with code X'4A'; otherwise, the associated data byte is written to static random access memory (SRAM) address X'02' as the pending duration (low) and S7 is set on to indicate that speaker parameters have been queued. (It is assumed that pending duration (high) and pending frequency have been previously set in SRAM addresses X'01', X'15', and X'16', respectively.)

If the speaker facilities are busy (S0 = 1 or S1 = 1), the command is accepted with a parameters queued acknowledgement. When the facilities become available, these queued parameters are validated and activated.

Otherwise, if speaker facilities are currently available, S7 is set off. The pending frequency and duration parameters are validated and, if not valid, an appropriate command reject code is returned. Valid speaker parameters are activated and the command is accepted with a speaker started acknowledgement.

(Refer to "Speaker Functional Description" on page 7-27.)

## Write to UART Device (X'03' and X'04')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'4B'	Command rejected – UART interface is disabled (M12 = 0)
X'4C'	Command rejected – adapter is busy with a previous write to UART (S4 = 1).



If the command is accepted, the associated data byte is transmitted to the UART (tablet) device.

The adapter has dual internal buffers for blocking reports received from a tablet device. Each buffer has a 6-byte length in order to receive a device report up to 6 bytes long. One buffer can be in the process of being transferred to the system while the other is being filled by a report from the device. If the device attempts to start a third report while the two buffers are busy, the third report (and all subsequent ones) are discarded until a buffer is available.

The M5 mode bit specifies whether blocking of received reports is active or inactive. When inactive (M5 = 0), a byte received from the device is passed on to the system. When blocking is active (M5 = 1), then report bytes received are buffered and blocked according to the blocking factor in SRAM address X'19'. Bytes of a blocked report are only transferred to the system when the complete report has been received by the adapter.

If the adapter detects a parity error on a report byte being received from the tablet, the adapter discards that report.

### **Write UART – Control (X'03')**

All UART device commands not having a defined response must be issued with the **write UART – control** command to the adapter.

### **Write UART – Query (X'04')**

All UART device commands having a defined response must be issued with the **write UART – query** command to the adapter. If a device command requiring a response is sent and no response is received within 25 milliseconds, an error has occurred. The adapter returns the device error code X'EA' to the system.

### **Set UART Baud Rate (X'05')**

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code of X'00' indicating acceptance of the command. The associated data byte is written to SRAM address X'1B' and is a counter value used to control the UART transmit and receive baud rate. The valid rate is determined by the UART device plugged. The default rate is 9600 bits per second.

The counter value in SRAM address X'1B' is related to the baud rate according to the following formula:

$$\text{Counter value} = 256 - \frac{\text{Osc}}{192 * \text{Baud}}$$

Where: *Counter value* = positive number rounded to an integer less than 256

*Osc* = 8051 oscillator frequency (Hz)

*Baud* = desired baud rate (bits per second).

For an assumed OSC value of 9.216 MHz, valid baud rates and corresponding counter values can be tabulated as follows:

Baud Rate	Counter (decimal)
24,000	254
9,600	251
4,800	246
2,400	236
1,200	216
600	176
300	96.



## Initialize UART Framing (X'06')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'4E'	Command rejected – invalid framing parameter.

If the command is accepted, the associated data byte is written to SRAM address X'19'. It is used to control the parity generation and checking for UART transmission and reception, and to specify the number of bytes in the report received from the UART device. The framing parameter is defined as follows:

Bits	Description
7	1 = odd parity (default); 0 = even parity
6–3	Must be 0's
2–0	Blocking factor; valid values are 2, 3, 4 (default), 5, or 6.

**Note:** The blocking factor must match the report length of the tablet device when blocking is active (M5 = 1).

## Set Speaker Duration – High Byte (X'07')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'4A'	Command rejected – Speaker queue full (S7 = 1).

If the command is accepted, the associated data byte is written to SRAM address X'01' as the pending speaker duration – high byte. Note that no validation of the duration value is performed until the speaker parameters are activated.

(Refer to "Speaker Functional Description" on page 7-27.)

## Set Speaker Frequency – High and Low Byte (X'08' and X'09')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'4A'	Command rejected – speaker queue full (S7 = 1).

If the command is accepted, the associated data byte is written to SRAM address X'15' for high byte command X'08', or SRAM address X'16' for low byte command X'09', as the pending speaker frequency. Note that no validation of the frequency value is performed until the speaker parameters are activated.

(Refer to "Speaker Functional Description" on page 7-27.)



## Diagnostic Write Keyboard Port Pins (X'0C')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'51'	Command rejected – illegal mode (M10 = 0).

This command is executed only in diagnostic mode. Data byte bit 7 is written to the keyboard clock line and data byte bit 6 is written to the keyboard data line. These values are kept on the indicated keyboard interface lines until the interface is cleared or a subsequent diagnostic write keyboard port pin alters them.

## Write Shared RAM (X'1R')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code of X'00' indicating acceptance of the command. The associated data byte is written to the SRAM address specified by the low 4 bits of the command byte, which address the read or write shared RAM addresses X'00' through X'0F'.

## Extended Command Descriptions

The following section describes the extended commands for the keyboard tablet speaker adapter.

### Read Shared RAM (X'00' – X'1F')

The adapter acknowledges the command by returning an interrupt ID of 3 and an associated data byte containing the contents of the SRAM address specified by bits 4–0 of the extended command byte. The command allows system software to read any single byte of read, write, or read-only shared RAM addresses X'00' through X'1F'.

### Reset Mode Bit (X'20' – X'2F')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'51'	Command rejected – illegal mode (M10 = 0).

If the command is accepted, the designated mode bit in shared RAM is cleared (set to X'0'). The mode bit affected is determined by the decimal value of the extended command byte bits 3–0, for example, mode bits 0 through 15.

Resetting of mode bit M11 can only be done while in diagnostic mode (M10 = 1); otherwise, the command is rejected.

### Set Mode Bit (X'30' – X'3F')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code of X'00' indicating acceptance of the command. The designated mode bit in shared RAM is set to 1. The mode bit affected is determined by the decimal value of the extended command byte bits 3–0, for example, mode bits 0 through 15.



### **Initialize Speaker Volume (X'40' – X'43')**

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code of X'00' indicating acceptance of the command. The speaker volume controls, and the associated mode bits 8 and 9, are set according to the specific command:

<b>Code</b>	<b>Description</b>
<b>X'40'</b>	Speaker volume = Off; M9, M8 = 00
<b>X'41'</b>	Speaker volume = Low; M9, M8 = 01
<b>X'42'</b>	Speaker volume = Medium; M9, M8 = 10
<b>X'43'</b>	Speaker volume = High; M9, M8 = 11.

The volume controls remain set until altered by a subsequent **initialize speaker volume** command, a set or reset of mode bits 8 or 9, or an adapter reset.

Volume commands are immediately executed, for example, volume is not a queued parameter.

### **Terminate Speaker and Reset Duration (X'44')**

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating the current state of the speaker:

<b>Code</b>	<b>Description</b>
<b>X'02'</b>	Speaker was already inactive (S1 = 0); no action taken
<b>X'03'</b>	Speaker was active and has been terminated.

Speaker termination clears Status bits S0 and S1, the speaker duration ticks in SRAM addresses X'13' and X'14', and clears the adapter speaker frequency input signal (allowing other features to control the speaker frequency).

Status bit S7 is set off, indicating no speaker command queued. Any values in SRAM addresses X'01', X'02', X'15', and X'16' are not affected.

### **Set Scan Count for System Attention Keystroke Sequence (X'5S')**

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

<b>Code</b>	<b>Description</b>
<b>X'00'</b>	Command accepted
<b>X'50'</b>	Command rejected – invalid count specified.

If the command is accepted, the indicated scan count (S) is set in SRAM address X'17' and the state of the system attention keystroke sequence is reset. The extended command byte bits 3–0 are interpreted as a scan count for the system attention keystroke sequence. Valid values for S are: 1, 2, or 3 (default).

### **Execute 8051 Soft Reset (X'60')**

If the adapter is not in the diagnostic mode (that is, if M10 = 0), the command is rejected by returning an interrupt ID of 0 with an informational code of X'51': Illegal Mode.

If the adapter is in the diagnostic mode (M10 = 1), an abnormal end condition is forced. An interrupt ID of 7 with an abnormal end code of X'A0' is returned to the system, plus two additional bytes, each with an interrupt ID of 7. The adapter then re-initializes the 8051 chip, performs the self-tests, and reports the self-test completion code with an interrupt ID of 6. At the conclusion of this command, the adapter is in normal operations with SRAM set to the



defined defaults and the device interfaces cleared. (Refer to “Abnormal End Codes” on page 7-34.)

### **Force System Attention Interrupt (X'62')**

If the adapter is not in the diagnostic mode (that is, if M10 = 0), the command is rejected by returning an interrupt ID of 0 with an informational code of X'51': Illegal Mode.

If the adapter is in diagnostic mode (M10 = 1), an immediate system attention interrupt is forced. This command performs the identical function initiated by the system attention special keystroke sequence except that no final scan code is queued nor placed in SRAM address X'1A'. This command ends by returning an interrupt ID of 0 with an informational code of X'00'.

### **Diagnostic Sense Keyboard and UART Port Pins (X'70')**

If the adapter is not in diagnostic mode (that is, if M10 = 0), the command is rejected by returning an interrupt ID of 0 with an informational code of X'51': Illegal Mode.

If the adapter is in diagnostic mode (M10 = 1), an interrupt ID of 3 is returned with an associated data byte defined as follows:

Bit	Description
0	State of UART 'Receive Data' (RXD) input signal
1	0
2	State of 'Keyboard Clock In' input signal
3	0
4	0
5	State of 'Keyboard Data In' input signal
6	0
7	0.

### **Dump Adapter Shared RAM (X'80' and X'81')**

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'60'	Command rejected – RAM queue is busy with prior dump command.

If the command is accepted, a 16-bit block of shared RAM is queued for transmission to the system. The command X'80' queues the read or write SRAM addresses X'00' – X'0F'; the command X'81' queues the read-only SRAM addresses X'10' – X'1F'. When the queued block is ready for transmission to the system, an interrupt ID of 4 with an associated data byte containing the byte-count (16) is returned. Then each SRAM byte is returned, in sequence, with an interrupt ID of 3.

Subsequent **dump adapter SRAM** or **dump RAS logs** commands are rejected until the currently queued block has been transmitted.



## Dump RAS Logs With or Without Reset (X'82' and X'83')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code indicating acceptance or rejection of the command:

Code	Description
X'00'	Command accepted
X'60'	Command rejected – RAM queue is busy with prior dump command.

If the command is accepted, a 12-byte block of RAS Logs information (SRAM addresses X'20' – '2B') is queued for transmission to the system. When the queued block is ready for transmission to the system, an interrupt ID of 4 with an associated data byte containing the byte-count (12) is returned. Then each SRAM byte is returned, in sequence, with an interrupt ID of 3. After the block has been transmitted, and if the operation was initiated with command X'82', then SRAM addresses X'20' – '2B' is zeroed out. Otherwise, SRAM is not affected.

Subsequent **dump adapter SRAM** or **dump RAS Logs** commands are rejected until the currently queued block has been transmitted.

## Restore Initial Conditions (X'90')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code of X'00' indicating completion of the command. Shared RAM addresses X'00' through X'1B' are initialized to the defined default conditions. The speaker command queue is cleared and any active speaker operation is terminated. All queued keyboard and tablet input data is cleared and the respective interface controls reset to initial conditions.

This command does not affect RAS Logs or the attached keyboard and tablet devices. The 8051 self-tests are not performed. This command does not clear certain types of information that may be, or have been, queued for transmission to the host (for example, error code, status report informational code, or requested RAM block).

## Read 8051 Release Marker (X'E0' – X'EF')

The adapter acknowledges the command by returning an interrupt ID of 3 and an associated data byte containing the contents of the selected Release Marker byte in 8051 ROM. The particular byte is specified by bits 3–0 of the extended command byte. Each command allows system software to read the corresponding single byte of the 16-byte Release Marker field defined as follows:

Bytes	Description
0 – 1	VV: Version number of 8051 code (2-character ASCII)
2 – 7	MMDDYY: Date of 8051 Version, Month, Date and Year (6-character ASCII)
8 – 11	SSSS: Unique serial number of 8051 chip (32-bit binary number)
12 – 13	KK: Check sum on serial number field (16-bit binary number)
14 – 15	ZZ: Check sum on entire 8051 ROM (16-bit binary number).

## NOP (X'F0' – X'FF')

The adapter acknowledges receipt of the command with an interrupt ID of 0 and an informational code of X'00' indicating acceptance of the command. The low 4 bits of the extended command byte are ignored. This command performs no other function.

## Functional Description and Allocation Map

(Refer to "8051 RAM Allocation" on page 7-40.)



## Read-Shared RAM

SRAM Address	Read or Write Shared RAM Data	Number of Bytes	Initialized Value
00	Keyboard acknowledge byte	1	FA
01-02	Pending speaker duration ticks	2	0000
03	Keyboard resend command	1	FE
04	Keyboard break code	1	F0
05	Maximum retry count before a keyboard hard error is reported	1	08
06	Keyboard echo command	1	EE
07	Click duration	1	36
08	Click suppress scan code number 1	1	12
09	Click suppress scan code number 2	1	59
0A	Click suppress scan code number 3	1	39
0B-0C	Click frequency – high or low	2	0896
0D	Keystroke initiate system attention – scan number 1. Also, click suppress scan code number 4.	1	11
0E	Keystroke initiate system attention – scan number 2. Also, click suppress scan code number 5.	1	19
0F	Keystroke initiate system attention – scan number 3. (The default is “any scan match.”)	1	FF



## Modes and Status Bits in Shared RAM

SRAM Address /Bit	M/S Bit #	Adapter Mode and Status Bits	Initialized Value
10.0	M0	Report receipt of keyboard acknowledgement byte with an informational interrupt	0 = No
10.1	M1	Report completion of UART transmit with an informational interrupt	0 = No
10.2	M2	[Unassigned]	
10.3	M3	Report completion of speaker tone with an informational interrupt	1 = Yes
10.4	M4	[Unassigned]	0
10.5	M5	Blocking of received UART bytes is active.	1 = Yes
10.6	M6	System attention keystroke SEQ search is active.	1 = Yes
10.7	M7	Suppress click for defined scan code set.	0 = No
11.0	M8	Speaker volume bit 0	0 = Med
11.1	M9	Speaker volume bit 1	1 = Med
11.2	M10	Diagnostic mode is in effect.	0 = No
11.3	M11	Keyboard interface is enabled with clear.	1 = Yes
11.4	M12	UART interface is enabled with clear.	0 = No
11.5	M13	[Unassigned]	
11.6	M14	Inhibit keystroke auto-click	1 = Yes
11.7	M15	Ignore UART input (manufacturing test only)	0 = No
12.0	S0	Speaker frequency timer busy	0
12.1	S1	Timeout timer busy	0
12.2	S2	[Unassigned]	0
12.3	S3	Keyboard transmit is busy.	0
12.4	S4	UART transmit is busy.	0
12.5	S5	Click busy	0
12.6	S6	[Unassigned]	0
12.7	S7	Speaker queue full	0

### Notes:

1. Mode bits can be altered using the **set mode bit** and **reset mode bit** command. Status bits are read-only. Mode bits 0–3 enable the corresponding bits in the status report informational codes (refer to this section on page 7-34).
2. Setting mode bit 11 or 12 clears the keyboard or UART interface, respectively, even if the mode bit was already set. Mode bit 11 can be cleared only in diagnostic mode. The keyboard interface should not normally be disabled.
3. Mode bit 15 = 1 causes the adapter to discard all data received by the UART port.



## Read-Only Shared RAM

SRAM Address	Read or Write Shared RAM Data	Number of Bytes	Initialized Value
13–14	Active speaker duration ticks remaining.	2	0000
15–16	Pending speaker frequency – high/low	2	0000
17	Scan count for system attention keystroke sequence (maximum # of scan codes is 3).	1	03
18	Keyboard sequence state High 4 bits = system attention sequence state	1	00
19	UART framing: MSB = odd/even parity control (1 = odd) Low 3 bits = blocking factor (valid: 2 to 6)	1	84
1A	System attention scan code – actual third byte received in a 3-byte sequence.	1	00
1B	UART baud rate (counter reload value) defaults to 9600 bps. See note 3.	1	FB
1C	Actual 8051 self-test completion code (value indicated is for the “good machine”. See note 2.	1	AE
1D–1E	8051 abnormal end information	2	0000
1F	Error code for most recent interrupt ID of 7. See note 1.	1	00

### Notes:

1. Refer to “Adapter Error Codes Returned to System” on page 7-34.
2. Refer to “Adapter Self-Test After a Power-On, System, or Adapter Reset Operation” on page 7-32.
3. Refer to “Set UART Baud Rate” on page 7-17.



## RAS Logs in Shared RAM

SRAM Address	Read-Only Shared RAM – RAS Logs	Number of Bytes
20–21	Number of keyboard frames received divide by 16 (not a keystroke count)	2
22–23	Number of keyboard receive retries performed (includes those resulting in hard errors)	2
24	Number of keyboard receive hard errors	1
25	Number of keyboard frames transmitted	1
26	Number of keyboard transmit retries performed (includes those resulting in hard errors)	1
27	Number of keyboard transmit hard errors	1
28–29	Number of UART frames (bytes) received divided by 16.	2
2A	Number of UART frames transmitted	1
2B	Number of UART receive errors (number of frames received with bad parity)	1

### Notes:

1. Logs are read-only by way of the **dump RAS logs** commands. All counters are zeroed upon completion of the **dump RAS logs with reset** command or an 8051 reset operation.
2. Each counter is an 8-bit or 16-bit binary value. Due to the large number of keyboard or UART frames received, those two counters are incremented only once per 16 frames received.



---

## Adapter Speaker Control

The speaker resides in the system keyboard. It is completely controlled and powered by the adapter board. The speaker interface on the Standard I/O Board contains circuits to control the volume of the frequency signal. Both the adapter and Micro Channel audio signal have access to the speaker interface.

The adapter implements the logical speaker functions used by system software, including frequency, duration, and volume parameters. The speaker functions are command-driven and queued. The adapter also implements a keystroke click tone for the keyboard.

## Sharing of Speaker Input With the Micro Channel Audio Signal

The Micro Channel audio signal activates the speaker through the interface circuits on the Standard I/O Board. This is an audio voltage sum node. This signal is ORed with the signal from the adapter. The quiescent state of the signal is a high voltage level (logical X'1'). Either the adapter or audio signal can alter it. No form of interlocking is provided to prevent one source from driving the speaker signal while the other source is driving it.

## Speaker Frequency Control

The speaker frequency is set by system software. Commands allow for a 2-byte count value to be sent to the adapter. The adapter uses the count value to initialize internal counters, which control a speaker frequency signal driving the adapter input to the speaker interface OR circuit. System software sends the count value to the adapter using the set speaker frequency commands with high-byte and low-byte values sent separately.

The permissible range of the count value provides a speaker input frequency range from a maximum of 12019 Hz to a minimum of 23 Hz.

The 2-byte count value used by the adapter is related to the generated speaker frequency according to the following formulas:

$$N = 11 - \log \left( 9216000 * \frac{F}{OSC} \right)$$

$$Cx = 2 \exp[ N ]$$

$$Ct = \frac{OSC / (24 * F) - K0}{Cx} - K1$$

Where:

$Cx$	=	Count value, high byte
$Ct$	=	Count value, low byte
$\log$	=	Base 2 logarithm function
$F$	=	Desired speaker frequency (Hz)
$OSC$	=	8051 oscillator frequency (Hz)
$[N]$	=	The exponent on the number 2 where the brackets mean that $N$ is to be rounded to an integer and if $[N] > 0$ , then use $[N]$ , otherwise, use 0.
$K0$	=	3.7 (a constant)
$K1$	=	9.25 (another constant).

Limitations:

$Cx$	=	Element of the set: (1, 2, 4, 8, 16, 32, 64)
$Ct$	=	Integer in the range [19, 255] for $Cx = 1$ , the range [120, 255] if otherwise

Exception:

A  $Cx$  value of 0 provides a silent tone. See the following note.



Example:

$F$  = 2500 Hz = Desired speaker frequency  
 $OSC$  = 9.216 MHz = Assumed 8051 oscillator  
 $N$  = -0.29, so  $[N]$  = 0  
 $Cx$  = 1  
 $Ct$  = 141

For the assumed  $OSC$  frequency of 9.216 MHz, the formulas for  $Cx$  and  $Ct$  can be simplified and tabulated as follows:

Frequency Range (Hz)	$Cx$ (decimal)	$Ct$ Formula
23 – 45	64	$(6000 / F) - 9.31$
46 – 90	32	$(12000 / F) - 9.37$
91 – 181	16	$(24000 / F) - 9.48$
182 – 362	8	$(48000 / F) - 9.71$
363 – 724	4	$(96000 / F) - 10.18$
725 – 1432	2	$(192000 / F) - 11.10$
1433 – 12019	1	$(384000 / F) - 12.95$

**Note:** A timed period of silence can be obtained by setting the frequency count value to 0. Specifically, if the frequency – high byte  $Cx = 0$ , the low byte  $Ct$  is ignored. The speaker operation is executed exactly as for any valid frequency, except that the frequency signal pin from the 8051 chip to the speaker is held in its quiescent state.

## Speaker Duration Control

A speaker tone duration is specified by the system as the number of duration ticks where each duration tick is 1/128 of a second. Duration ticks are specified as a 15-bit number with a permissible range from 0 to 32767, for example, X'0000' through X'7FFF'. The actual number of duration ticks is one greater than the number specified, providing a tone duration from 7.8 milliseconds to 256 seconds. System software must first specify the tone duration–high byte, then issue the write to speaker command specifying the duration–low byte.

## Speaker Volume Control

The adapter controls the speaker volume for both the adapter use of the speaker and the Micro Channel adapter use. System software must ensure that only one source is driving the frequency control. In either case, system software must use the adapter function to specify the speaker volume level.

Four levels of volume are available: off, low, medium, and high. The adapter initializes itself to medium after a reset operation. System software can alter this setting at any time. The adapter retains the last setting in the 8051 shared RAM mode bits 8 and 9 as follows:

M9, 8	Volume
B'00'	Off
B'01'	Low
B'10'	Medium (default)
B'11'	High



## Speaker Command Queue Description

Execution of the **write to speaker** command is queued if the speaker facilities are busy doing a click or a previous speaker command. Speaker commands are rejected if a Write to Speaker command is currently queued, for example, if the queue consists of one set of parameters (frequency and duration). When speaker parameters are written to the adapter, they are initially pending (queued). They become active when speaker facilities are available.

A keystroke click occurs only if speaker facilities are immediately available, for example, the click is not queued.

Speaker volume is assumed to be a global parameter under user control. As such, it is not queued. Speaker volume commands are immediately executed.

## Functional Operation

Commands to set frequency (high and low bytes) and duration (high byte) place these parameters in the queue as pending. The commands are rejected if the queue is full.

The **write to speaker** command places the duration (low byte) parameter in the queue as pending, with an implied request to activate the speaker using the pending parameters. The command is rejected if the queue is full.

Normal response to the **write to speaker** command is either speaker started or parameters queued.

## Implementation

Speaker command queueing uses the following adapter resources:

- SRAM address X'01–02' = pending duration high and low.

(Active duration is maintained in SRAM address X'13–14'.)

- SRAM address X'15–16' = pending frequency high/low.

(Active frequency kept in private RAM.)

- Status bit S7 defined as speaker queue full
- Status report informational code bit 3; set on if speaker parameters (frequency or duration) are invalid:
  - When a keystroke click occurs (frequency only)
  - OR
  - When the pending speaker command is dequeued.

**Note:** Parameters are not validated until actually activated.

## Keystroke Click Description

The keyboard defined in the referenced specification does provide a mechanical acoustical feedback for a key button being depressed. The adapter can also provide a keystroke auto-click function that generates a click tone whenever a valid scan code is received. The resulting clicks only represent a replacement for a mechanically generated keyboard click. The adapter-generated click only means that the adapter has received a valid scan code and queued it to the system. The click cannot be interpreted to mean that the system software has received the keystroke.



## Functional Operation

The keystroke auto-click function defaults to inactive. It can be enabled by setting mode bit M14 off. The click defaults to a 301 Hz tone for 1.68 msec. The click frequency and duration can be respecified by system software.

A keystroke click is initiated only when the adapter receives the make of any valid scan code and queues it to the system. Valid scan codes are those in the range X'01' through X'9F'.

If the byte received from the keyboard is a break code, then the next byte received does not initiate a click.

If the keystroke queue is full when a keyboard byte is received, then the adapter replaces the last byte in the queue with the overrun code X'00', discards the current byte received, and does not initiate a click.

## Implementation

The keystroke auto-click function uses the following adapter resources:

- SRAM address X'07' = click duration
- SRAM address X'0B–0C' = click frequency
- Mode Bit M14 defined as inhibit keystroke auto-click
- Status report informational code bit 3 is set on if the click frequency in SRAM is not valid when the keystroke auto-click function is initiated.

The click duration is specified by SRAM address X'07' as a count of 30.52 microsecond ticks. The count in SRAM is an 8-bit number. The actual number of duration ticks is one greater than the number specified in SRAM, providing a range of click durations from 30.52 microsecond to 7.8 millisecond in 30.52 microsecond increments.

The click frequency is specified by SRAM addresses X'0B' – '0C'. These two bytes are used as the frequency counter values Cx and Ct, respectively. They are interpreted by the adapter exactly as for any other speaker frequency. They must adhere to the same rules as specified in the section "Speaker Frequency Control" on page 7-27. Cx = 0 is not valid, in other words, click cannot be the silent tone.

If an click frequency that is not valid is written to SRAM and the adapter attempts to click due to a keystroke, no click occurs. An unsolicited status report informational code byte with bit 3 on is posted to the system.

## Click Suppression for Defined Scan Code Set

The keystroke auto-click function can be suppressed for a definable set of five scan codes by setting mode bit M7 = '1'. The default set of scan codes is defined by the five SRAM addresses X'08 thru 0A', '0D', and '0E'. (Note that the last two locations also define the system attention interrupt sequence.) A scan code can be deleted from the suppress set by setting its corresponding SRAM location to X'FF'.

## Click Interference with other Speaker Operations

If the adapter's speaker facilities are busy with a previous speaker command when a click is to be initiated, then no click occurs. It is ignored. Conversely, if the speaker is busy with a click, and the system issues a **write to speaker** command, then the parameters for that command are queued and activated when the click completes.

The adapter has no knowledge of a Micro Channel device using the speaker. Consequently, if the Micro Channel device is using the speaker and the adapter issues a click, the sound observed is a combination the Micro Channel device-generated tone and the click tone.



---

## Adapter RAS and Security Functions

The following sections describe the RAS and security functions provided for by the adapter.

### Detection of Special Keystroke Sequences

As keystrokes pass from the keyboard to the system through the adapter, the adapter searches for a special sequences of scan codes. The sequence causes the adapter to initiate a system attention interrupt. The scan code received by the adapter is always presented to the system with an interrupt ID of 1.

### Initiate System Attention Interrupt

The adapter is able to force the system attention interrupt signal to the system. This signal is initiated by a unique 3-keystroke sequence. The system attention interrupt default keystroke sequence can be redefined or disabled by the system software.

Incoming keystrokes are searched by the adapter for the special sequence whose default is as follows:

1. Scan Code X'11'.
2. Scan Code X'19'.
3. Match on any scan code other than X'62'.

Key position requirements: first two keys must be make or break.

When the sequence is detected, an immediate system attention interrupt is initiated. The sequence search can be disabled by the system software setting mode bit M6 off. When enabled, the sequence search is always active (except when the keylock switch is on, or if system software has disabled the keyboard interface with M11 = 0, or has disabled keyboard scanning). The system can alter the keystroke sequence searched, and the length of the sequence can be one, two, or three keystrokes.

Read or write SRAM defines the three scan codes. If the sequence length is 2 or 3, then the first two scan codes can appear in either order. If the sequence length is 3 and scan number is 3 in SRAM = X'FF' (default conditions), then the sequence detection is satisfied when any third scan code is received (other than a break code). The third scan code received of a length-3 sequence is always placed in read-only SRAM address X'1A' before the system attention interrupt is initiated by the adapter.

Individual activity counters maintain counts of frames received and transmitted through the keyboard and tablet UART ports. Retry counters accumulate counts of keyboard retries performed. Individual error counters accumulate the number of hard errors for the keyboard and tablet UART ports.

The keyboard and speaker ports can be diagnostically wrapped or sensed at the signal points that leave the system board. No external wrap connectors are required. Also, UART port signals can be diagnostically sensed. The 8051 chip performs a self-test function after a system reset. Self-tests can also be performed on command from the system.

The adapter initiates retry operations with the keyboard on transmit and receive errors. The number of retries performed can be redefined by system software. The adapter returns 8051-detected error conditions to the system where additional error recovery procedures can be performed.



## Diagnostic Wraps

Certain adapter port signals can be diagnostically sensed through the following mechanisms:

- Extended command X'70':= ( refer to page 7-21.)
- Reading 8255 PB Port: = (refer to page 7-10.)

Adapter command byte bit 5 = 1 causes the UART 'TxD' signal to be wrapped to the UART 'RxD' signal on the system board at the 8051 pins. To wrap a byte, use the **write UART – control (X'23')** adapter command with any desired data byte. That data byte is then read at the UART RxD pin and posted to the system with an interrupt ID of 2.

## Adapter Self–Test After a Power–On, System, or Adapter Reset Operation

The following 8051 facilities are tested:

- 8051 ROM checksum
- 8051 RAM
- 8051 internal registers.

As the tests in this self–test series are executed, a bit–significant completion code is generated. The code is initialized to X'51' and, as each test completes, the corresponding bit is complemented if the test was successful. Tests are executed and bits are complemented from most– to least–significant with bit meanings as follows:

Bit	Description	Meaning
7	Reset initiation indicator	1 = initialized by a hardware reset *
6	Accumulator and PSW test	0 = test passed OK
5	ROM checksum test	1 = test passed OK
4	RAM test with X'AA' data	0 = test passed OK
3	RAM test with X'55' data	1 = test passed OK
2	RAM test – addressing	1 = test passed OK
1	RAM test with X'00' data	1 = test passed OK
0	Control registers checksum test	0 = test passed OK

Thus, if all tests pass successfully, the resultant completion code is X'AE'. The completion code is stored in read–only shared RAM address X'1C' and posted to the system in the 8255 chip with an interrupt ID of 6.

\* A hardware reset is initiated by a system power–on, system reset, or adapter reset operation. If bit 7 = 0, then the self–test was initiated by the 8051 chip having forced an abnormal end to the normal operations. (Refer to "Abnormal End Codes" on page 7-34.)

## Diagnose Functions Executed on System Command

- Execute 8051 soft reset (extended command X'60'); force abnormal end code X'A0', perform self–tests, and report the completion code with an interrupt ID of 6
- Force system attention interrupt (extended command X'62').

## Adapter Informational Codes Returned to System

The data byte associated with interrupt ID of 0 is an informational code. Informational codes are classified as:



- Acknowledgement
- Command reject
- Status report.

### Acknowledgement Informational Codes

One of the following codes is returned as the data byte associated with interrupt ID of 0 to acknowledge receipt of an adapter command from the system.

<b>X'00'</b>	Host command acknowledged.
<b>X'01'</b>	Speaker started.
<b>X'02'</b>	Speaker inactive.
<b>X'03'</b>	Speaker terminated.
<b>X'04'</b>	Speaker parameters queued.

### Command Reject Informational Codes

If the adapter rejects a command from the system, it does so by returning one of the following codes as the data byte associated with interrupt ID of 0.

<b>Reject Code</b>	<b>Command Handler Issuing</b>
<b>X'41'</b> = Reject keyboard transmit busy	Write keyboard operation
<b>X'43'</b> = Reject keyboard disabled	Write keyboard operation
<b>X'44'</b> = Reject invalid keyboard data	Write keyboard operation
<b>X'47'</b> = Reject invalid speaker duration	Write keyboard operation
<b>X'48'</b> = Reject invalid speaker freq	Write keyboard operation
<b>X'4A'</b> = Reject speaker queue full	Write keyboard operation
<b>X'4B'</b> = Reject UART disabled	Write UART operation
<b>X'4C'</b> = Reject UART transmit busy	Write UART operation
<b>X'4D'</b> = Reject invalid baud	Set UART baud rate operation
<b>X'4E'</b> = Reject invalid framing	Set UART baud rate operation
<b>X'50'</b> = Reject invalid count	Set sequence A length operation
<b>X'51'</b> = Reject illegal mode	Diagnostic and diagnostic sense operation
<b>X'60'</b> = Reject ram queue busy	Dump RAM block operation
<b>X'7F'</b> = Reject undefined operation	



## Status Report Informational Codes

The adapter can send an unsolicited status report informational code to the system with interrupt ID of 0. A status report code is distinguished from acknowledgement and command reject codes by having the most significant bit of the byte set on. The remaining bits of the status report byte are bit-significant. The bits are defined as follows:

Bit	Condition
7 = 1 (Status report identifier)	
6 = Speaker tone completed	Conditioned by M3=1
5 = Keyboard returned Ack	Conditioned by M0=1
4 = Unassigned	
3 = Invalid speaker parameter	(Click or queued frequency or duration)
2 = UART transmit complete	Conditioned by M1=1
1 = RAS log near overflow	
0 = RAS log overflowed.	

## Adapter Error Codes Returned to System

The data byte associated with interrupt ID of 7 is an error code. Error codes are classified as:

- Abnormal end codes
- Device error codes.

### Abnormal End Codes

An abnormal end code occurs when the adapter has detected an unrecoverable error condition and forced an abnormal end to normal operations. The data byte (abnormal end code) provides the nature of the specific condition detected. When the abnormal end code has been read by the system, 2 additional bytes are provided by the adapter with an interrupt ID of 7 that define the 8051 microcode address, which detected the condition. After the microcode address has been transferred to the system, or if the 8051 chip times out waiting for the system, the 8051 chip re-initializes itself and attempts to restore normal operations. The 8051 shared RAM is reset to its default state. After the soft reset self-test has been performed by the 8051 chip, the completion code is reported with an interrupt ID of 6. Note that the high order bit of the completion code is off, indicating a soft reset of the 8051 chip rather than a hardware-initiated reset.

### System Action Required

When the system detects an abnormal end code, it should then prepare to receive the subsequent two codes (8051 address). That information should be logged as an incident. Re-initialize the 8051 shared RAM, if any of the defaults had been previously altered. If an abnormal end code immediately re-occurs, the system should issue an adapter reset operation.

#### Codes

X'A0'	=	Diagnose initiated 8051 soft reset
X'A1'	=	Word queue low decode
X'A3'	=	Host transmit queue decode
X'A4'	=	Increment RAS log decode
X'A6'	=	Wild branch
X'A7'	=	System reset failed.



## Device Error Codes

A device error code occurs when an unexpected condition has been detected by the 8051 microcode at a device interface. The problem may not be with the device itself. The adapter attempts to continue normal operations.

### System Action Required

The device error code information should be logged as an incident. The system may have to issue some kind of device reset command to try and clear the condition. A particular device error code may suggest a recovery procedure. If the condition persists, the system should issue an adapter reset operation.

### Codes

#### **X'E0' = Keyboard Transmit Timeout**

Adapter has started to transmit a frame to the keyboard. Transmission of that frame did not complete within the maximum allowed time. The keyboard interface has been cleared and re-enabled. The keyboard **echo** command could be issued to test the circuits to the keyboard and back. If the condition continues, the keyboard may have been unplugged.

#### **X'E1' = Keyboard Receive Timeout**

The adapter has started to receive a frame from the keyboard. Reception of that frame did not complete within the maximum allowed time. The keyboard interface has been cleared and re-enabled if mode bit 11 = 1. The keyboard **echo** command could be issued to test the circuits to the keyboard and back. If the condition continues, the keyboard may have been unplugged.

#### **X'E2' = Kbd Ack Not Received**

An acknowledgement response was expected from the last transmission to the keyboard, but something other than an acknowledgement was received. The actual keyboard response byte has been queued to the system.

#### **X'E3' = Unexpected Kbd Ack Received**

An unexpected acknowledgement response was received from the keyboard. It was unexpected because a prior transmission had not been performed by the adapter.

#### **X'E4' = Hard Error on Kbd Frame Receive**

The adapter has unsuccessfully performed the maximum number of keyboard frame receive retries. The frame received has a solid framing error. The keyboard interface is enabled for further communication.

#### **X'E5' = Hard Error on Kbd Frame Transmit**

The adapter has unsuccessfully performed the maximum number of keyboard frame transmit retries. The keyboard has responded with a solid error condition (resend) due to the keyboard receiving an **invalid frame** or **invalid** command. The keyboard interface is enabled for further communication.

#### **X'E6' = Kbd Clock Pin Not Plus**

See the following explanation of the X'E7' code.

#### **X'E7' = Kbd Clock Pin Not Minus**

The adapter has attempted to release or hold the clock signal to the keyboard. The read-back of the signal on the clock line did not verify. If this condition persists, it may indicate a interface circuit failure. It may also be due to noise on the cable, or the keyboard may have been unplugged. The adapter attempts to continue the operation.



**X'E8' = UART Interrupt Without TIRI**

A serial port interrupt occurred without a transmit or receive identifier. The interrupt is ignored and processing continues.

**X'E9' = UART Transmit Timeout**

The 8051 serial port buffer was loaded for transmission to the UART device. Transmission did not complete within the maximum allowed time. Processing continues. A **UART device wrap** command could be issued. If the condition is still persists, the UART device may have been unplugged or the 8051 serial port failed.

**X'EA' = UART Ack Timeout**

The 8051 serial port did transmit the byte to the UART device. The device did not respond within the maximum allowed time. Processing continues. Other device commands could be issued. If no response is received, the device may have been unplugged.



---

## Keyboard Device Support Notes

The information in this section supplements that in the respective keyboard section of this manual.

### Keyboard Commands

#### Resend (X'FE')

The system should not issue this command. The function is handled internally by the adapter retry facility.

#### Echo (X'EE')

The adapter passes the keyboard response to the system.

### Keyboard Outputs

#### Resend (X'FE')

The adapter intercepts this output and handles it in the adapter retry facility. The X'FE' is not passed on to the system.

#### Ack (X'FA')

The adapter intercepts this output and reports the occurrence of the acknowledgement only if mode bit 0 = 1.

#### Overrun (X'00')

The adapter also injects an overrun response if the keystrokes fill the adapter's 5-byte FIFO queue.

---

## Adapter Design Notes

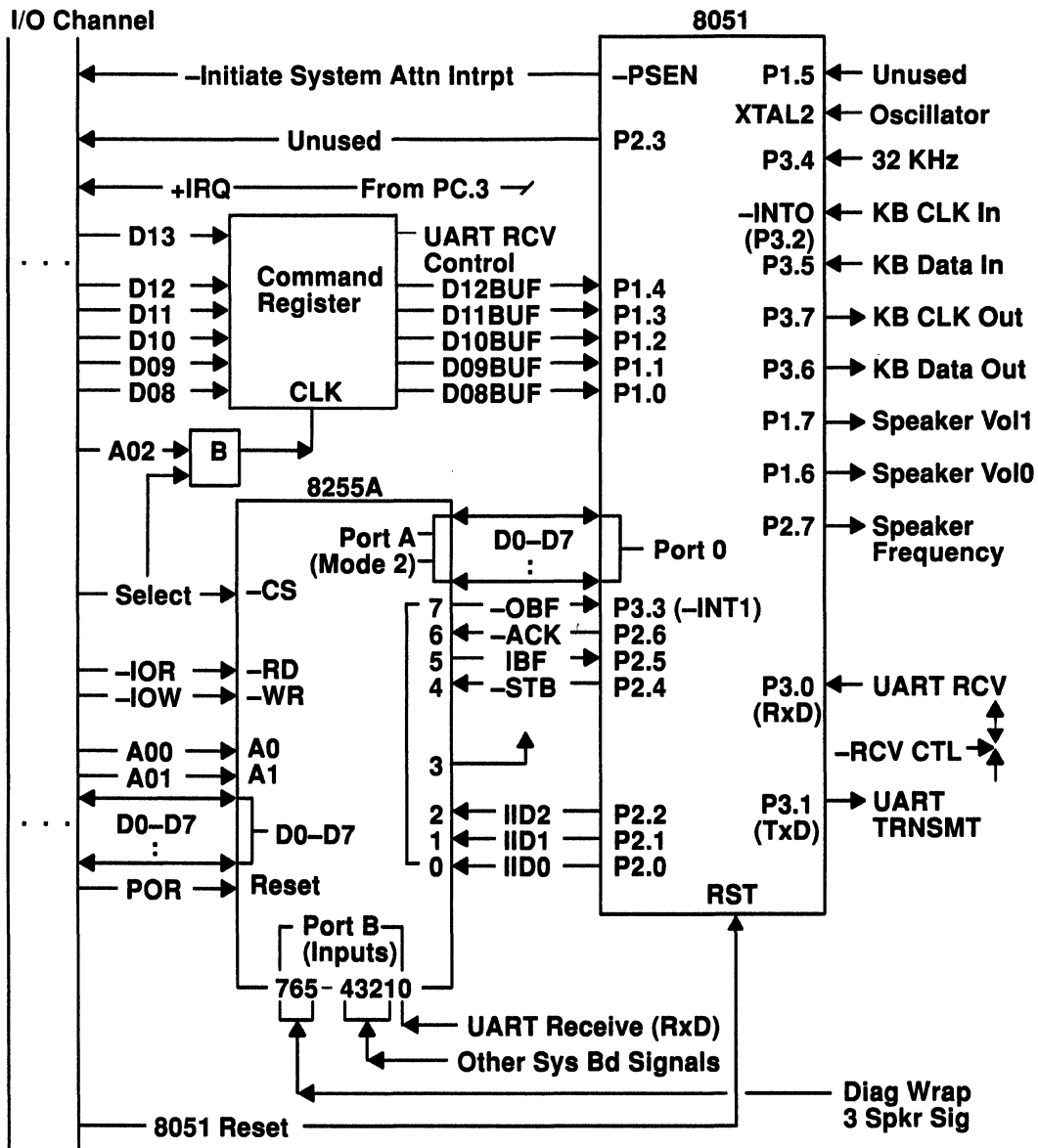
Figure 88 shows the adapter components and the system interface.

Figure 89 shows the interface logic for keyboard and speaker signals between the 8051 chip and the connector to the keyboard.

Figure 90 shows the interface logic for tablet signals between the 8051 UART and the connector to the tablet.

Also shown in the Interface figures are the points where signals are wrapped to the 8255 Port B inputs for diagnostic sensing.





**Box B: Decode the Command Register Select = (NOT A02) and (I/O Write) and (Adapter Select)**

Figure 88. Adapter Logic and System Interface



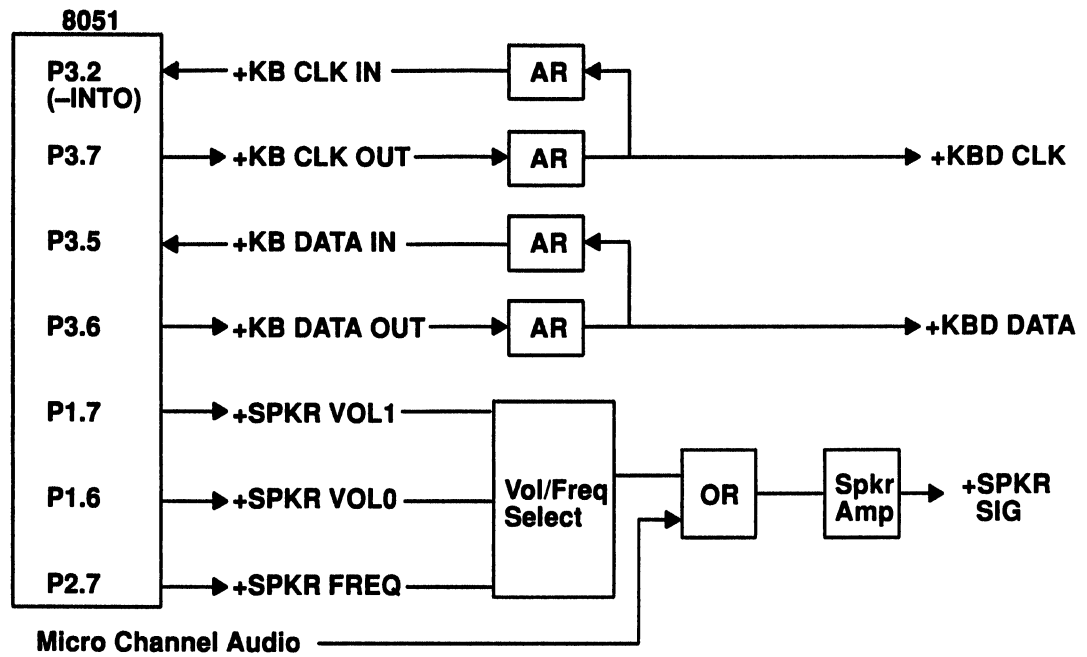


Figure 89. Adapter-to-Keyboard Connector Interface

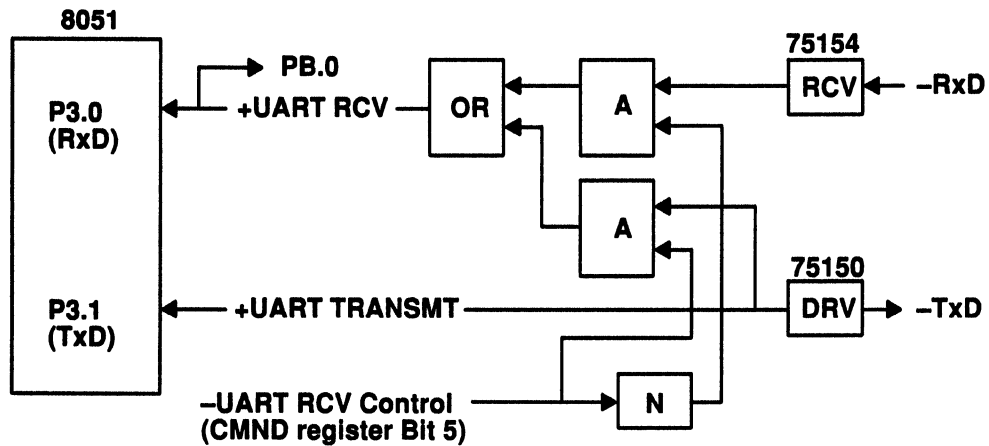


Figure 90. Adapter-to-Tablet (UART Port) Connector Interface



## Channel I/O Device Address Bit Decoding

Read or Write Standard I/O Keyboard Adapter

	MSB		LSB
	1111 11		1
	5432 1098 7654 3210		
I/O Address:	0000 0000 0101 0RPP		= X'005—'
Where:	PP = 8255 Port addressing for I/O Read and Write		
	PP	IOR	IOW
	00:	Read PA register	Write PA register
	01:	Read PB register	Write PB register
	10:	Read PC register	Write PC register
	11:	Illegal	Write Control

PA is defined to be the adapter data register.  
It sends and receives the low-order data byte.

R = Adapter command register control

R	IOR	IOW
1:	No action	No Action
0:	No action	Latch high byte in adapter register

**Note:** A normal 2-byte write operation to the adapter would have RPP = 000, to I/O address X'0050'. This loads the command and data bytes in their respective registers.

## 8051 RAM Allocation

Absolute RAM Address	Shared RAM Address	Usage	Number of Bytes
00–07		Register Bank 0	8
08–0F		Register Bank 1	8
10–17		Register Bank 2	8
18–1F		Register Bank 3	8
20–2B		Work Area	12
2C–2E	10–12	24–Bit Shared Bit Space	3
2F–3B	13–1F	Shared (R.O.) Byte Space	13
3C–4B	00–0F	Shared (R/W) Byte Space	16
4C–57	20–2B	RAS Logs	12
58–63		UART Current Receive Blocks	12
64–68		Keystroke Queue	5
69–7F		Stack	23
Total:			128



---

## Adapter and Keyboard Initialization Procedure

The following are the recommended steps for initializing the adapter and keyboard after a system reset operation occurs:

1. Activate adapter reset.

The 8051 reset is activated by the power-on operation, system reset, or system software directly activating the bit in the CRR register. Additionally, the power-on operation causes a power-on reset (POR) to the 8255 chip.

2. Configure 8255 chip.

Write the adapter Config 8255 Chip operation to properly configure the 8255 chip for communication with the 8051 chip.

3. Enable Host Interrupt IRQ

Write the adapter Enable IRQ operation to allow the interrupt request line to the system to be activated when the adapter is initiating a transfer to the system.

**Note:** This step can be done at some later time if the interrupts cannot be handled yet. Data transfers in subsequent steps are either interrupt-driven or polled.

4. Variable Delay Td.

Allows the keyboard to startup and perform its internal tests, quiescent clock, and data lines before the 8051 interface to the keyboard is active. The delay Td is determined by the relation:

$$Tkp + Td \geq Tkg + Tkb + Tkc$$

OR

$$Td \geq (Tkg - Tkp) + Tkb + Tkc$$

Where:

$Tkp$  = Time since keyboard power applied  
(usually same as time since system POR)

$Tkg$  = Time for keyboard to startup  
= 2 seconds (maximum) per keyboard specification

$Tkb$  = Time for keyboard basic assurance test (BAT) to run  
= 300 milliseconds to 500 milliseconds per keyboard specification

$Tkc$  = Time for keyboard to transmit completion  
< 20 milliseconds.

If the time since system POR is greater than about 2.5 seconds, no delay is necessary.

5. Release adapter reset.

Releasing the 8051 reset in CRR allows the 8051 chip to run its self-test and initialization.

6. Wait for adapter initialization response.

After the 8051 self-test and initialization is completed, the completion code is posted to the 8255 chip. This should occur within 100 milliseconds of the preceding release.



7. Validate adapter self-test completion code.

The expected machine data value is X'AE' with interrupt ID of 6.

8. Reset the keyboard.

Issue the adapter command to reset the keyboard. The expected adapter command response is X'00' with an interrupt ID of 0.

9. Wait for keyboard initialization response.

Wait for a data byte from the keyboard, which should be received within 500 milliseconds.

**Note:** If the byte returned is a device error code, for example, X'E0', the keyboard is probably not attached (unplugged).

10. Validate keyboard reset.

The data byte should have been returned with an interrupt ID of 1. Data byte should be as follows:

Keyboard basic assurance test (BAT) completion code = X'AA'.

The adapter and keyboard are now initialized to their defined defaults.



---

## Standard I/O Adapter Board to Device Interface

The following section shows the the device interfaces to the Standard I/O adapter board.

### Keyboard Port Interface

The following figure shows the keyboard port (receptacle) on the Standard I/O adapter board.

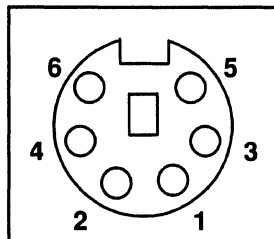


Figure 91. Keyboard Connector

Pin	Signal
1	Keyboard Data
2	Speaker Signal
3	Ground
4	+5 V dc
5	Keyboard Clock
6	Speaker Ground

### Tablet (UART Port) Device Interface

The following figure shows the tablet connector (receptacle) on the UART port of the Standard I/O adapter board.

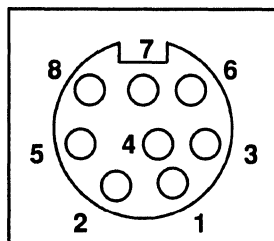


Figure 92. Tablet Connector

Pin	Signal
1	Ground
2	Direct Current (DC) Return (Ground)
3	+5 V dc
4	Reserved
5	Receive from device
6	Transmit to device
7	Reserved
8	Reserved







---

# Chapter 8. Keyboard

## Chapter Contents

Description .....	8-3
Power-On Routine .....	8-4
Power-On Reset .....	8-4
Basic Assurance Test .....	8-4
Sequential Key-Code Scanning .....	8-4
Buffer .....	8-4
Keys .....	8-4
Commands from the System .....	8-5
Commands to the System .....	8-6
Scan Codes .....	8-7
Set 1 Scan Code Tables .....	8-7
Set 2 Scan Code Tables .....	8-10
Set 3 Scan Code Tables .....	8-13
Clock And Data Signals .....	8-15
Data Stream .....	8-15
Data Output .....	8-16
Data Input .....	8-16
Keyboard Character Codes .....	8-17
Extended Code Functions .....	8-19
Shift Status .....	8-22
Shift Key Priorities and Combinations .....	8-22
Speaker .....	8-23
Key Position Layout .....	8-23
Keyboard Layouts .....	8-24
Cables and Connectors .....	8-29
Specifications .....	8-29
Power Requirements .....	8-29
Dimensions and Weight .....	8-29







---

## Description

The following features are contained on the keyboard used with the RISC System/6000 unit :

- Multikey buffer with overrun detection
- Num Lock, Scroll Lock, and Caps Lock LED indicators
- Multiple scan code sets
- A speaker to generate tones.

The keyboard interface is bidirectional and allows scan-coded outputs to be output from the keyboard and serial command, and data to be input from the RISC System/6000 unit.

The keyboard uses *membrane* technology, which relies upon a microcomputer to perform the keyboard scan function.

System software is allowed to have maximum flexibility in defining certain keyboard operations by having the keyboard return a unique scan code for each key. Scan codes are shown in the "Scan Codes" section on page 8-7. All keys are individually programmable to recognize the following conditions:

- Make only
- Make/Break
- Repeat
- Repeat Make/Break.

**Note:** Repeat and repeat make/break keys cannot be mixed on any keyboard.

The repeat rate is programmable from 2 to 30 characters per second. The repeat delay is programmable from 250 milliseconds to 1 second. A break code is formed by prefixing the scan code with X'F0'.

The microcomputer in the keyboard performs several functions, including a basic assurance test (BAT) at startup or when requested by the system. The following functions are also contained on the keyboard:

- Keyboard scanning
- Buffering up to three key scan codes
- Maintaining bidirectional serial communications with the system unit
- Executing the handshake protocol required by each scan code transfer.

The U.S. keyboard contains 101 keys arranged in four major groups (102 keys for other countries). The keys on the central portion of the keyboard are arranged in a standard typewriter layout. At the top are 12 user-defined function keys. To the immediate right of the central portion are 10 cursor-control keys and on the far right is a numeric pad. The three LED indicators are mounted on the upper right corner of the keyboard. The keyboard key positions and layout are shown in "Key Position Layout" on page 8-24 and "Keyboard Layouts" on page 8-23.

The keyboard has two tilt positions for operator comfort (7 and 15 degrees) and a speaker for use by the system for generating tones.



---

## Power-On Routine

The activities described in the following sections take place when power is first applied to the keyboard.

### Power-On Reset

The keyboard logic generates a power-on reset function (POR) when power is first applied to the keyboard. POR takes a minimum of 150 milliseconds and a maximum of 2 seconds from the time power is first applied to the keyboard.

### Basic Assurance Test

The basic assurance test (BAT) consists of a keyboard processor test, a checksum of the read-only memory (ROM), and a random-access memory (RAM) test. During the BAT, activity on the clock and data lines is ignored. The LED indicators are turned on at the beginning and off at the end of the BAT. The BAT takes a minimum of 300 milliseconds and a maximum of 500 milliseconds. This is in addition to the time required by the POR.

On satisfactory completion of the BAT, a completion code (X'AA') is sent to the system, and keyboard scanning begins. If a BAT error occurs, the keyboard sends an error code to the system. The keyboard is then disabled pending command input. Completion codes are sent between 450 milliseconds and 2.5 seconds after POR, and between 300 and 500 milliseconds after a reset command is acknowledged.

Immediately following POR, the keyboard monitors the signals on the keyboard 'clock' and 'data' lines and sets the line protocol.

---

## Sequential Key-Code Scanning

The keyboard detects all keys pressed and sends each scan code in correct sequence. When not being serviced by the system, the keyboard stores the scan codes in its buffer.

### Buffer

A 16-byte first-in-first-out (FIFO) in the keyboard stores the scan codes until the system is ready to receive them. A buffer-overflow condition occurs when more than 16 bytes are placed in the keyboard buffer. An overflow code replaces the 17th byte. If more keys are pressed before the system allows keyboard output, the additional data is lost.

When the keyboard is allowed to send data, the bytes in the buffer are sent as in normal operation, and new data entered is detected and sent. Response codes do not occupy a buffer position.

If keystrokes generate a multiple-byte sequence, the entire sequence must fit into the available buffer space, or the keystroke is discarded and a buffer-overflow condition occurs.

### Keys

The make scan of a key is sent to the keyboard controller when the key is pressed. When the key is released, its break scan code is sent.

When a key is pressed and held down, the keyboard sends the make code for that key, delays 500 milliseconds  $\pm$  20 percent, and again begins sending a make code for that key at a rate of 10.9 characters per second  $\pm$  20 percent. The typematic rate and delay can be modified by the X'F3' command (see "Commands from the System" that follows).



If two or more keys are held down, only the last key pressed repeats at the typematic rate. Typematic operation stops when the last key pressed is released, even if other keys are still held down. If a key is pressed and held down while keyboard transmission is inhibited, only the first make code is stored in the buffer. This prevents buffer overflow because of typematic action.

Scan code set 3 allows key types to be changed by the system. See "Set 3 Scan Code Table" on page 8-13 for the default settings.

---

## Commands from the System

The following command set is used by the keyboard:

### Hex

Command	Description
FF	Reset: Perform power-on check (BAT) and report.
FE	Resend: Resend last byte.
FD	Set make: Set designated keys to the make code.
FC	Set make/break: Set designated keys to make/break.
FB	Set repeat: Set designated keys to repeat.
FA	Set repeat make/break: Set all keys to repeat make/break.
F9	Set all make: Set all keys to make.
F8	Set all make/break: Set all keys to make/break.
F7	Set all repeat: Set all keys to repeat.
F6	Set default: Reinitializes the basic default conditions.
F5	Default disable: Set all keys to repeat make/break and discontinue scanning.
F4	Enable: Start scanning.
F3	Set repeat rate/delay: Set the repeat delay to $1 + \text{the binary value of bits 5 and 6 (bit 7 (MSB)=0)} \times 250\text{ms} \pm 20\%$ . Set the repeat rate according to the following: <ul style="list-style-type: none"> <li>• <math>\text{Period} = ((8 + A) \times 2)^B / 2 \times 0.00834 \text{ seconds} \pm 20\%</math>.</li> <li>• A = Binary value of bits 2, 1, 0 (LSB)</li> <li>• B = Binary value of bits 3, 4.</li> </ul>
F2	Read ID: Respond with X'83AB'.
F1, 00 to EC	Commands not valid: If these commands are sent, the keyboard returns a <b>resend</b> (X'FE') command.
F0	Select alternate scan code: Select scan code set 1, 2, or 3.
EF	Layout ID: Responds with X'B0' and X'BF' for the United States (U.S.) 101 keyboard layout or X'B1' and X'BF' for the World Trade (WT) 102 keyboard layout.
EE	Echo: Respond with X'EE'.



<b>ED</b>	Set LED indicator: The LED indicators are set according to bits 0, 1, 2 of the subsequent byte.	
	<b>Bit</b>	<b>LED</b>
	<b>0</b>	Scroll Lock
	<b>1</b>	Num Lock
	<b>2</b>	Caps Lock

---

## Commands to the System

The following codes are output by the keyboard:

**Hex**

<b>Output</b>	<b>Description</b>
<b>FF(or 00)</b>	Overflow or key detection error: Output by keyboard following keystroke buffer overflow or a nonidentified switch closure. X'FF' is output if scan code set 1 is being used, or X'00' is output if scan code set 2 or 3 is being used.
<b>FE</b>	Resend: Output by keyboard following receipt of input that is not valid or any input with incorrect parity.
<b>FC</b>	If an error occurs during the BAT, the keyboard sends this code, discontinues scanning, and waits for system response or POR to restart.
<b>FA</b>	Ack: Output by keyboard following receipt of any valid input other than the <b>echo</b> or <b>resend</b> commands.
<b>F0</b>	Break code: Prefixed to the scan code of a make/break key to indicate break of the key.
<b>EE</b>	Echo: Output in response to <b>echo</b> command.
<b>AA</b>	BAT completion code: Output following completion of the self-tests. Any other output indicates a test error.
<b>83AB</b>	Keyboard ID: Output in response to the <b>read id</b> command or following BAT completion code.



---

## Scan Codes

Three sets of scan codes are supported and are selectable by way of commands. Scan code set 2 is the default. The system changes the keyboard to Set 3 Scan Code.

### Set 1 Scan Code Tables

For scan code set 1, each key is assigned a base scan code and sometimes extra codes to generate artificial shift states in the system. The typematic scan codes are identical to the base scan code for each key.

Table 1 shows the code sent for the keys, regardless of any shift states in the keyboard or system. See "Key Position Layout" on page 8-23 and "Keyboard Layouts" on page 8-24 to determine the character associated with each key number.

Table 1 Set 1, Keyboard Scan Codes Page 1 of 2					
Key Number	Make Code	Break Code	Key No.	Make Code	Break Code
1	29	A9	49	2F	AF
2	02	82	50	30	B0
3	03	83	51	31	B1
4	04	84	52	32	B2
5	05	85	53	33	B3
6	06	86	54	34	B4
7	07	87	55	35	B5
8	08	88	56	73	F3
9	09	89	57	36	B6
10	0A	8A	58	1D	9D
11	0B	8B	60	38	B8
12	0C	8C	61	39	B9
13	0D	8D	62	E0 38	E0 B8
14	7D	FD	64	E0 1D	E0 9D
15	0E	8E	90	45	C5
16	0F	8F	91	47	C7
17	10	90	92	4B	CB
18	11	91	93	4F	CF
19	12	92	94	7C	FC
20	13	93	96	48	C8
21	14	94	97	4C	CC
22	15	95	98	50	D0
23	16	96	99	52	D2
24	17	97	100	37	B7



Table 1 Set 1, Keyboard Scan Codes Page 2 of 2					
Key Number	Make Code	Break Code	Key No.	Make Code	Break Code
25	18	98	101	49	C9
26	19	99	102	4D	CD
27	1A	9A	103	51	D1
28	1B	9B	104	53	D3
29	2B	AB	105	4A	CA
30	3A	BA	106	4E	CE
31	1E	9E	107	7E	FE
32	1F	9F	108	E0 1C	E0 9C
33	20	A0	109	78	F8
34	21	A1	110	01	81
35	22	A2	112	3B	BB
36	23	A3	113	3C	BC
37	24	A4	114	3D	BD
38	25	A5	115	3E	BE
39	26	A6	116	3F	BF
40	27	A7	117	40	C0
41	28	A8	118	41	C1
42	2B	AB	119	42	C2
43	1C	9C	120	43	C3
44	2A	AA	121	44	C4
45	56	D6	122	57	D7
46	2C	AC	123	58	D8
47	2D	AD	125	46	C6
48	2E	AE			



The remaining keys send a series of codes that are dependent on the various shift keys (Ctrl, Alt, and Shift), and the state of the Num Lock key (on or off). Because the base scan code is identical to another key, an extra code (X'E0' or X'E1') has been added to the base code to make it unique. The remaining keys (with the exception of the keypad/key , Prnt Sc/Sys Req, and Pause/Break keys) are shown in Table 2.

<b>Table 2 Set 1, Remaining Keys Scan Code</b>			
<b>Key No.</b>	<b>Shift + Num Lock Make/Break</b>	<b>Shift Make/Break</b>	<b>Num Lock Make/Break</b>
75	E0 52/	E0 AA E0 52/	E0 2A E0 52/
	E0 D2/	E0 D2 E0 2A	E0 D2 E0 AA
76	E0 53/	E0 AA E0 52/	E0 2A E0 52/
	E0 D2	E0 D2 E0 2A	E0 D3 E0 AA
79	E0 4B/	E0 AA E0 4B/	E0 2A E0 4B/
	E0 CB	E0 CB E0 2A	E0 CB E0 AA
80	E0 47/	E0 AA E0 47/	E0 2A E0 47/
	E0 C7	E0 C7 E0 2A	E0 C7 E0 AA
81	E0 4F/	E0 AA E0 47/	E0 2A E0 4F/
	E0 C7	E0 CF E0 2A	E0 CF E0 AA
83	E0 48/	E0 AA E0 48/	E0 2A E0 48/
	E0 C8	E0 C8 E0 2A	E0 C8 E0 AA
84	E0 50/	E0 AA E0 50/	E0 2A E0 50/
	E0 D0	E0 D0 E0 2A	E0 D0 E0 AA
85	E0 49/	E0 AA E0 50/	E0 2A E0 49/
	E0 C9	E0 C9 E0 2A	E0 C9 E0 AA
86	E0 51/	E0 AA E0 51/	E0 2A E0 51/
	E0 D1	E0 D1 E0 2A	E0 D1 E0 AA
89	E0 4D/	E0 AA E0 4D/	E0 CD E0 2A/
	E0 CD	E0 CD E0 2A	E0 CD E0 AA

**Note:** If the left Shift key is held down, the AA/2A Shift Make and Break are sent with the other scan codes. If the right Shift key is held down, B6/36 is sent. If both Shift keys are down, both sets of codes are sent with the other scan code.

The following describes the scan code for the keypad/key:

**Key Number    Make/Break**

**95**                      Scan code: E0 35/E0 B5

                            Shift: E0 AA E0 35/E0 B5 E0 2A.

**Note:** If the left Shift key is held down, the AA/2A shift make and break are sent with the other scan codes. If the right Shift key is held down, B6/36 is sent. If both Shift keys are down, both sets of codes are sent with the other scan code.



The following describes the scan code for the Prnt Sc/Sys Req key:

**Key Number    Make/Break**

**124**                    Scan code: E0 35/E0 B5  
                              Shift: E0 AA E0 35/E0 B5 E0 2A.

The following describes the scan code for the Pause/Break key:

**Key Number    Make/Break**

**126**                    Scan code: E0 2A E0 37/E0 B7 E0 AA  
                              Ctrl and Shift: E0 37/E0 B7  
                              Alt: 54/D4.

**Note:** This key is not typematic. All associated scan codes occur on the make of the key.

## Set 2 Scan Code Tables

For scan code set 2, each key is assigned a unique 8-bit make scan code that is sent when the key is pressed. Each key also sends a break code when the key is released. The break code consists of 2 bytes, the first of which is the break code prefix 'F0'; the second byte is the same as the make scan code for that key. The typematic scan code for a key is the same as the key make code.

Table 3 shows the codes sent for the keys, regardless of any shift states in the keyboard or system. See "Key Position Layout" on page 8-23 and "Keyboard Layouts" on page 8-24 to determine the character associated with each key number.

Table 3 Set 2, Keyboard Scan Codes Page 1 of 2					
Key No.	Make Code	Break Code	Key No.	Make Code	Break Code
1	0E	F0 0E	49	2A	F0 0E
2	16	F0 16	50	32	F0 32
3	1E	F0 1E	51	31	F0 31
4	26	F0 26	52	3A	F0 3A
5	25	F0 25	53	41	F0 41
6	2E	F0 2E	54	49	F0 49
7	36	F0 36	55	4A	F0 4A
8	3D	F0 3D	56	51	F0 51
9	3E	F0 3E	57	59	F0 59
10	46	F0 42	58	14	F0 14
11	45	F0 45	60	11	F0 11
12	4E	F0 4E	61	29	F0 29
13	55	F0 55	62	E0 11	E0 F0 11
14	6A	F0 6A	64	E0 14	E0 F0 14
15	66	F0 66	90	77	F0 77
16	0D	F0 0D	91	6C	F0 6C
17	15	F0 15	92	6B	F0 6B



Table 3 Set 2, Keyboard Scan Codes Page 2 of 2					
Key No.	Make Code	Break Code	Key No.	Make Code	Break Code
18	1D	F0 1D	93	69	F0 69
19	24	F0 24	94	68	F0 68
20	2D	F0 2D	96	75	F0 75
21	2C	F0 2C	97	73	F0 73
22	35	F0 35	98	72	F0 72
23	3C	F0 3C	99	70	F0 70
24	43	F0 43	100	7C	F0 7C
25	44	F0 25	101	7D	F0 7D
26	4D	F0 4D	102	74	F0 74
27	54	F0 54	103	7A	F0 7A
28	5B	F0 5B	104	71	F0 71
29	5D	F0 5D	105	7B	F0 7B
30	58	F0 58	106	79	F0 79
31	1C	F0 1C	107	6D	F0 6D
32	1F	F0 1F	108	E0 5A	E0 F0 5A
33	23	F0 33	109	63	F0 63
34	2B	F0 2B	110	76	F0 76
35	34	F0 34	112	05	F0 05
36	33	F0 33	113	06	F0 06
37	3B	F0 3B	114	04	F0 04
38	42	F0 42	115	0C	F0 0C
39	4B	F0 4B	116	03	F0 03
40	4C	F0 4C	117	0B	F0 0B
41	52	F0 52	118	83	F0 83
42	5D	F0 5D	119	0A	F0 0A
43	5A	F0 5A	120	01	F0 01
44	12	F0 12	121	09	F0 09
45	61	F0 61	122	78	F0 78
46	1A	F0 1A	123	07	F0 07
47	22	F0 22	125	7E	F0 7E
48	21	F0 21			



The remaining keys send a series of codes that are dependent on the state of the shift keys (Ctl, Alt, and Shift), and the state of NumLock (on or off). Because the base scan code is identical to another key, an extra code (X'E0') is added to the base code to make it unique. With the exception of the keypad / (95), PrtSc/SysRq (124), and Pause/Break (126) keys, the remaining keys are shown in Table 4.

<b>Table 4 Set 2, Remaining Keys Scan Code</b>			
<b>Key No.</b>	<b>Shift + Num Lock Make/Break</b>	<b>Shift Make/Break</b>	<b>Num Lock Make/Break</b>
75	E0 70/	E0 F0 12 E0 70/	E0 12 E0 70/
	E0 F0 70	E0 F0 70 E0 12	E0 F0 70 E0 F0 12
76	E0 71/	E0 F0 12 E0 71/	E0 12 E0 71/
	E0 F0 71	E0 F0 71 E0 12	E0 F0 71 E0 F0 12
79	E0 6B/	E0 F0 12 E0 6B/	E0 12 E0 6B/
	E0 F0 6B	E0 F0 6B E0 12	E0 F0 6B E0 F0 12
80	E0 6C/	E0 F0 12 E0 6C/	E0 12 E0 6C/
	E0 F0 6C	E0 F0 6C E0 12	E0 F0 6C E0 F0 12
81	E0 69/	E0 F0 12 E0 69/	E0 12 E0 69/
	E0 F0 69	E0 F0 69 E0 12	E0 F0 69 E0 F0 12
83	E0 75/	E0 F0 12 E0 75/	E0 12 E0 75/
	E0 F0 75	E0 F0 75 E0 12	E0 FO 75 E0 F0 12
84	E0 72/	E0 F0 12 E0 72/	E0 12 E0 72/
	E0 F0 72	E0 F0 72 E0 12	E0 F0 72 E0 F0 12
85	E0 7D/	E0 F0 12 E0 7D/	E0 12 E0 7D/
	E0 F0 7D	E0 F0 7D E0 12	E0 F0 7D E0 F0 12
86	E0 7A/	E0 F0 12 E0 7A/	E0 12 E0 7A/
	E0 F0 7A	E0 F0 7A E0 12	E0 F0 7A E0 F0 12
89	E0 74/	E0 F0 12 E0 74/	E0 12 F0 74/
	E0 F0 74	E0 F0 12 E0 74	E0 F0 74 E0 F0 12

**Note:** If the left Shift key is held down, the F0 12/12 shift make and break are sent with the other scan codes. If the right Shift key is held down, F0 59/59 is sent. If both Shift keys are down, both sets of codes are sent with the other scan code. This applies to key 95 also.

The following describes the scan code for the keypad / key:

**Key Number    Make/Break**

**95**                      Scan code: E0 4A/E0 F0 4A  
                               Shift: E0 F0 12 E0 4A/E0 F0 4A E0 12.

The following describes the scan code for the SysReq key:

**Key Number    Make/Break**

**124**                     Scan code: E0 12 E0 7C/E0 F0 7C E0 F0 12



Ctrl and Shift: E0 7C/E0 F0 7C

Alt: 84/F084.

The scan code for the Pause/Break key is as follows:

**Key No.            Make/Break**

**126                Make code: E1 14 77 E1 F0 14 F0 77**

**Ctrl: E0 7E E0 F0 7E.**

**Note:** This key is not typematic. All associated scan codes occur on the make of the key.

### Set 3 Scan Code Tables

For scan code set 3, each key is assigned a unique 8-bit make scan code that is sent when the key is pressed. Each key also sends a break code when the key is released. The break code consists of 2 bytes: the break code prefix X'F0' and the make scan code for the key. The typematic scan code for a key is the same as the key make code. With this scan code set, each key sends only one scan code, and no key is affected by the state of any other key.

Table 5 shows the code sent for the keys, regardless of any shift states in the keyboard or system. See "Key Position Layout" on page 8-23 and "Keyboard Layouts" on page 8-24 to determine the character associated with each key number.

**Note:** Default key state is typematic.

Table 5 Set 3, Keyboard Scan Codes Page 1 of 3					
Key No.	Make Code	Break Code	Key No.	Make Code	Break Code
1	0E	F0 0E	55	4A	F0 4A
2	16	F0 16	56	51	F0 51
3	1E	F0 1E	57	59	F0 59
4	26	F0 26	58	11	F0 11
5	25	F0 25	60	19	F0 19
6	2E	F0 2E	61	29	F0 29
7	36	F0 36	62	39	F0 39
8	3D	F0 3D	64	58	F0 58
9	3E	F0 3E	75	67	F0 67
10	46	F0 46	76	64	F0 64
11	45	F0 45	79	61	F0 61
12	4E	F0 4E	80	6E	F0 6E
13	55	F0 55	81	65	F0 65
14	5D	F0 5D	83	63	F0 63
15	66	F0 66	84	60	F0 61
16	0D	F0 0D	85	6F	F0 6F
17	15	F0 15	86	6D	F0 6D
18	1D	F0 1D	89	6A	F0 6A



Table 5 Set 3, Keyboard Scan Codes Page 2 of 3					
Key No.	Make Code	Break Code	Key No.	Make Code	Break Code
19	24	F0 24	90	76	F0 76
20	2D	F0 2D	91	6C	F0 6C
21	2C	F0 2C	92	6B	F0 6B
22	35	F0 35	93	69	F0 69
23	3C	F0 3C	94	68	F0 68
24	43	F0 43	95	77	F0 77
25	44	F0 44	96	75	F0 75
26	4D	F0 4D	97	73	F0 73
27	54	F0 4D	98	72	F0 72
28	5B	F0 5B	99	70	F0 70
29	4C	F0 5C	100	7E	F0 7E
30	14	F0 14	101	7D	F0 7D
31	1C	F0 1C	102	74	F0 74
32	1B	F0 1B	103	7A	F0 7A
33	23	F0 23	104	71	F0 71
34	2B	F0 2B	105	84	F0 84
35	34	F0 34	106	7C	F0 7C
36	33	F0 33	107	7B	F0 7B
37	3B	F0 3B	108	79	F0 79
38	42	F0 42	109	78	F0 78
39	4B	F0 4B	110	08	F0 08
40	4C	F0 4C	112	07	F0 07
41	52	F0 52	113	0F	F0 0F
42	5A	F0 5A	114	17	F0 17
43	5A	F0 5A	115	1F	F0 1F
44	12	F0 12	116	27	F0 27
45	13	F0 13	117	27	F0 27
46	1A	F0 1A	118	37	F0 37
47	22	F0 22	119	3F	F0 3F
48	21	F0 21	120	47	F0 47
49	2A	F0 2A	121	4F	F0 4F
50	32	F0 32	122	56	F0 56
51	31	F0 31	123	5E	F0 5E
52	3A	F0 3A	124	57	F0 57



Table 5 Set 3, Keyboard Scan Codes Page 3 of 3					
Key No.	Make Code	Break Code	Key No.	Make Code	Break Code
53	41	F0 41	125	5F	F0 5F
54	49	F0 49	126	62	F0 62

## Clock And Data Signals

The keyboard and system communicate over the clock and data lines. The source of each of these lines is an open-collector device on the keyboard that allows either the keyboard or system to force a line to an inactive (low) level. When no communication is occurring, the clock line is an active (high) level. The state of the data line is held active (high) by the keyboard.

When the system sends data to the keyboard, it forces the data line to an inactive level and allows the clock to go to an active level.

An inactive signal has a value of at least 0, but not more than +0.7 volts. A signal at the inactive level is a logical 0. An active signal has a value of at least +2.4, but not more than +5.5 volts. A signal at the active level is a logical 1. Voltages are measured between a signal source and the DC network ground.

When the keyboard sends data to or receives data from the system, it generates the 'clock' signal to time the data. The system can prevent the keyboard from sending data by forcing the 'clock' line to an inactive level. The 'data' line can be active or inactive during this time.

During the BAT, the keyboard allows the clock and data lines to go to an active level.

## Data Stream

Data transmissions to and from the keyboard consist of an 11-bit data stream (mode 2) sent serially over the 'data' line. The following describes the keyboard data stream bits:

Bit	Description
11	Stop bit (always 1)
10	Parity bit (odd parity)
9	Data bit 7 (most significant bit)
8	Data bit 6
7	Data bit 5
6	Data bit 4
5	Data bit 3
4	Data bit 2
3	Data bit 1
2	Data bit 0 (least significant bit)
1	Start bit (always 0)

The Parity bit is either 1 or 0, and the 8 Data bits, plus the Parity bit, always have an odd number of 1s.



**Note:** Mode 1 is a 9-bit data stream that does not have a Parity bit or Stop bit, and the Start bit is always 1.

## Data Output

When the keyboard is ready to send data, it first checks for a keyboard-inhibit or system request-to-send status on the clock and data lines. When the clock line is inactive, data is stored in the keyboard buffer. If the clock line is active and the data line is inactive (request-to-send), data is stored in the keyboard buffer, and the keyboard receives system data.

If the clock and data lines are both active, the keyboard sends the 0 Start bit, 8 Data bits, the Parity bit, and the Stop bit. Data is valid before the trailing edge and beyond the leading edge of the clock pulse. During transmission, the keyboard checks the clock line for an active level every 60  $\mu$ sec. When the system lowers the clock line from an active level after the keyboard starts sending data, a condition known as *line contention* occurs, and the keyboard stops sending data. If line contention occurs before the leading edge of the 10th clock signal (Parity bit), the keyboard buffer returns the clock and data lines to an active level. If line contention does not occur by the 10th clock signal, the keyboard completes the transmission. Following line contention, the system may request the keyboard to resend the data. Following a transmission, the system can inhibit the keyboard until the system processes the input, or until it requests that a response be sent.

## Data Input

When the system is ready to send data to the keyboard, it first checks to see if the keyboard is sending data. If the keyboard is sending data, but has not reached the 10th clock signal, the system can override the keyboard output by forcing the keyboard clock line to an inactive level. If the keyboard transmission is beyond the 10th clock signal, the system must receive the transmission.

If the keyboard is not sending, or if the system elects to override the keyboard output, the system forces the keyboard clock line to an inactive level for more than 60  $\mu$ sec. while preparing to send data. When the system is ready to send the Start bit (the 'data' is inactive), it allows the clock line to go to an active state.

The keyboard checks the state of the clock line at intervals of no more than 10 milliseconds. If a system 'request-to-send' signal is detected, the keyboard clocks in 11 bits. After the 10th clock, the keyboard checks for an active level on the data line, and if the line is active, it forces it inactive, and clocks once more. This action signals the system that the keyboard has received data. On receipt of this signal, the system returns to a ready state, in which it can accept keyboard output, or the system goes to the inhibited state until it is ready.

If the keyboard data line is at an inactive level following the 10th clock signal, a framing error has occurred, and the keyboard continues to clock until the data line becomes active. The keyboard then makes the data inactive and sends a **resend** command.

Each system command or data transmission to the keyboard requires a response from the keyboard before the system can send its next output. The keyboard responds within 20 ms unless the system prevents keyboard output. If the keyboard response is not valid or has a parity error, the system sends the command or data again. If F3 (set typematic rate/delay), F0 (select alternate scan codes), or ED (set/reset mode indicators) have been sent and acknowledged, and the value byte has been sent but the response is not valid or has a parity error, the system resends both the command and the value byte.



# Keyboard Character Codes

Keyboard Character Codes Page 1 of 3				
Key	Base Case	Uppercase	Ctrl	Alt
1	'	~	-1	(*)
2 Shift (Left)	1		-1	(*)
3	2	@	Null(000)(*)	(*)
4	3	#	-1	(*)
5	4	\$	-1	(*)
6	5	%	-1	(*)
7	6	^	RS(030)	(*)
8	7	&	-1	(*)
9	8	*	-1	(*)
10	9	(	-1	(*)
11	0	)	-1	(*)
12	-	-	US(031)	(*)
13	=	+	-1	(*)
15	Backspace (008)	Backspace (008)	Del(127)	(*)
16	→ (009)	←(*)	(*)	(*)
17	q	Q	DC1(017)	(*)
18	w	W	ETB(023)	(*)
19	e	E	ENQ(005)	(*)
20	r	R	DC2(018)	(*)
21	t	T	DC4(020)	(*)
22	y	Y	EM(025)	(*)
23	u	U	NAK(021)	(*)
24	i	I	HT(009)	(*)
25	o	O	SI(015)	(*)
26	p	P	DLE(016)	(*)
27	[	{	Esc(027)	(*)
28	]	}	GS(029)	(*)
29	\		FS(028)	(*)
30 Caps Lock	-1	-1	-1	-1
31	a	A	SOH(001)	(*)
(*) Refer to "Extended Code Functions".				



Keyboard Character Codes Page 2 of 3				
Key	Base Case	Uppercase	Ctrl	Alt
32	s	S	DC3(019)	(*)
33	d	D	EOT(004)	(*)
34	f	F	ACK(006)	(*)
35	g	G	BEL(007)	(*)
36	h	H	BS(008)	(*)
37	j	J	LF(010)	(*)
38	k	K	VT(011)	(*)
39	l	L	FF(012)	(*)
40	;	:	-1	(*)
41	'	"	-1	(*)
43	CR(013)	CR(013)	LF(010)	(*)
44 Shift (Left)	-1	-1	-1	-1
46	z	Z	SUB(026)	(*)
47	x	X	CAN(024)	(*)
48	c	C	ETX(003)	(*)
49	v	V	SYN(022)	(*)
50	b	B	STX(002)	(*)
51	n	N	SO(014)	(*)
52	m	M	CR(013)	(*)
53	,	<	-1	(*)
54	.	>	-1	(*)
55	/	?	-1	(*)
57 Shift (Right)	-1	-1	-1	-1
58 Ctrl (Left)	-1	-1	-1	-1
60 Alt (Left)	-1	-1	-1	-1
61	Space	Space	Space	Space
62 Alt (Right)	-1	-1	-1	-1
64 Ctrl/Alt (Right)	-1	-1	-1	-1
90 Num Lock	-1	-1	-1	-1
95	/	/	(*)	(*)
(*) Refer to "Extended Code Functions".				



Keyboard Character Codes Page 3 of 3				
Key	Base Case	Uppercase	Ctrl	Alt
100	*	*	(*)	(*)
105	—	—	(*)	(*)
106	+	+	(*)	(*)
108	Enter	Enter	LF(010)	(*)
110	Esc	Esc	Esc	(*)
112 through 123	Null(*)	Null(*)	Null(*)	Null(*)
125	—1	—1	—1	—1
126	Pause	Pause	Break	Break
(*) Refer to “Extended Code Functions”.				

Keys that have meaning only in Num Lock, Shift, or Ctrl states are described in “Special Character Codes that follows”.

Special Character Codes				
Key	Num Lock	Base Case	Alt	Ctrl
91	7	Home (*)	—1	Clear Screen
92	4	←(*)	—1	Reverse Word(*)
93	1	End (*)	—1	Erase to EOL(*)
96	8	↑(*)	—1	(*)
97	5	(*)	—1	(*)
98	2	↓(*)	—1	(*)
99	0	Ins	—1	(*)
101	9	Page Up (*)	—1	Top of Text and Home
102	6	→(*)	—1	Advance Word(*)
103	3	Page Down (*)	—1	Erase to EOS(*)
104	.	Delete (*)		
105	—	Sys Request	—1	—1
106	+	+ (*)	—1	—1
(*) Refer to “Extended Code Functions”.				

## Extended Code Functions

For certain functions that cannot be represented by a standard ASCII code, an extended code is used. A character code of 000 (null) is returned in the AL register.

This indicates that the system or application program should examine a second code, which indicates the actual function. Usually, this second code is the scan code of the primary key that was pressed. This code is returned in the AH register.

The extended codes are shown in the following table:



<b>Keyboard Extended Functions Page 1 of 2</b>	
<b>Second Code</b>	<b>Function</b>
1	Alt Esc
3	Null Character
14	Alt Backspace
15	←(Back-tab)
16–25	Alt Q, W, E, R, T, Y, U, I, O, P
26–28	Alt [ ] ↵
30–38	Alt A, S, D, F, G, H, J, K, L
39–41	Alt ; ‘ ’
43	Alt \
44–50	Alt Z, X, C, V, B, N, M
51–53	Alt , . /
55	Alt Keypad *
59–68	F1 to F10 Function Keys (Base Case)
71	Home
72	↑(Cursor Up)
73	Page Up
74	Alt Keypad –
75	←(Cursor Left)
76	Center Cursor
77	→(Cursor Right)
78	Alt Keypad –
79	End
80	↓(Cursor Down)
81	Page Down
82	Ins (Insert)
83	Del (Delete)
84–93	Shift F1 to F10
94–103	Ctrl F1 to F10
104–113	Alt F1 to F10
114	Ctrl PrtSc (Start/Stop Echo to Printer)
115	Ctrl ←(Reverse Word)
116	Ctrl →(Advance Word)
117	Ctrl End (Erase to End of Line—EOL)
118	Ctrl PgDn (Erase to End of Screen—EOS)



<b>Second Code</b>	<b>Function</b>
119	Ctrl Home (Clear Screen and Home)
120–131	Alt 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, –, = keys 2 through 13
132	Ctrl PgUp (Top 25 Lines of Text and Cursor Home)
133, 134	F11, F12
135, 136	Shift F11, F12
137, 138	Ctrl F11, F12
139, 140	Alt F11, F12
141	Ctrl Up/8
142	Ctrl Keypad –
143	Ctrl Keypad 5
144	Ctrl Keypad +
145	Ctrl Down/2
146	Ctrl Ins/0
147	Ctrl Del/.
148	Ctrl Tab
149	Ctrl Keypad /
150	Ctrl Keypad *
151	Alt Home
152	Alt Up
153	Alt Page Up
155	Alt Left
157	Alt Right
159	Alt End
160	Alt Down
161	Alt Page Down
162	Alt Insert
163	Alt Delete
164	Alt Keypad /
165	Alt Tab
166	Alt Enter



---

## Shift Status

- Shift:** This key temporarily shifts keys 1 through 13, 16 through 29, 31 through 41, and 46 through 55 to uppercase (base case if in Caps Lock state). Also, the Shift key temporarily reverses the Num Lock or non-Num Lock state of keys 91 through 93, 96, 98, 99, and 101 through 104.
- Ctrl:** This key temporarily shifts keys 3, 7, 12, 15 through 29, 31 through 39, 43, 46 through 52, 75 through 89, 91 through 93, 95 through 108, 112 through 124, and 126 to the Ctrl state.
- Alt:** This key temporarily shifts keys 1 through 29, 31 through 43, 46 through 55, 75 through 89, 95, 100, and 105 through 124 to the Alt state.

The Alt key also allows the user to enter any character code from 1 to 255. The user holds down the Alt key and types the decimal value of the characters desired on the numeric keypad (keys 91 through 93, 96 through 99, and 101 through 103). The Alt key is then released. If the number is greater than 255, a modulo-256 value is used. This value is interpreted as a character code and is sent through the keyboard routine to the system or application program. The Alt key is handled internally in the keyboard routine.

**Caps Lock:** This key shifts keys 17 through 26, 31 through 39, and 46 through 52 to uppercase. When the Caps Lock key is pressed again, it reverses the action. The Caps Lock key is handled internally in the keyboard routine. When the Caps Lock key is pressed, the Caps Lock mode indicator lights if the caps lock mode is entered.

**Scroll Lock:** When interpreted by appropriate application programs, this key indicates that the cursor control keys cause windowing over the text rather than moving the cursor. When the Scroll Lock key is pressed again, it reverses the action. The keyboard routine simply records the current shift state of the Scroll Lock key. It is the responsibility of the application program to perform the function. When the Scroll Lock key is pressed the scroll lock mode indicator lights if the scroll lock mode is entered.

**Num Lock:** This key shifts keys 91 through 93, 96 through 99, and 101 through 104 to uppercase. When the Num Lock key is pressed again, the action is reversed. Num Lock is handled internally to the keyboard routine. When the Num Lock key is pressed, the Num Lock mode indicator lights if num lock mode is entered.

## Shift Key Priorities and Combinations

If combinations of the Alt, Ctrl, and Shift keys are pressed and only one is valid, the following shows the priority:

1. Alt key
2. Ctrl key
3. Shift key.

The only valid combination is Alt and Ctrl, which is used in the system-reset function.



## Speaker

The keyboard has a 50 millimeter paper cone speaker located in the bottom cover. The speaker is rated at 200 milliwatts continuous power and has an 8 ohm  $\pm 15\%$  coil impedance at 1000. The speaker should have a frequency response of at least 500 to 5000 Hz  $\pm 20$  dB and should be driven exclusively by the system by way of two lines connected directly at the cable connector.

## Key Position Layout

Keyboard key position layouts are shown in Figure 93 and Figure 94.

[illegible]

**Figure 93. US 101 Key Position Layout**

[illegible]

**Figure 94. WT102 Key Position Layout**



# Keyboard Layouts

Keyboard layouts are in alphabetic order on the following pages. Nomenclature is on both the top and front face of the keys.

## Belgium

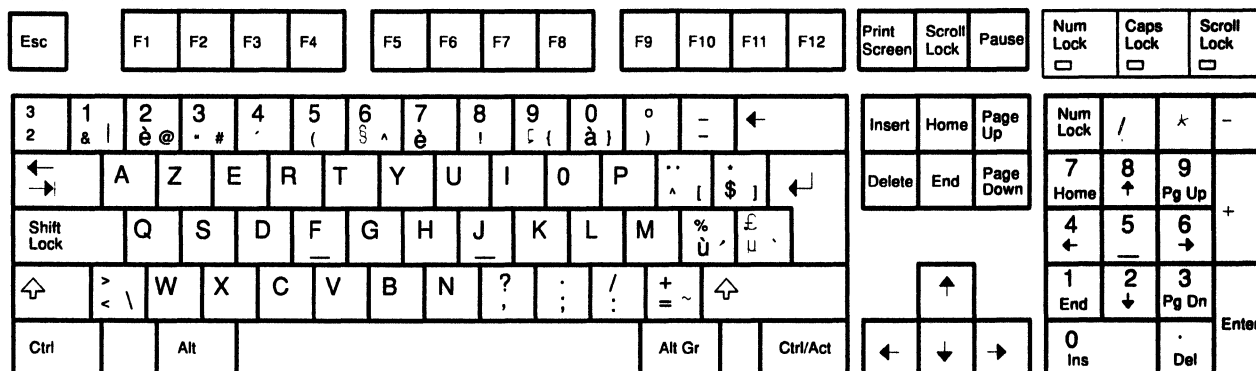


Figure 95. Belgian

## Canada

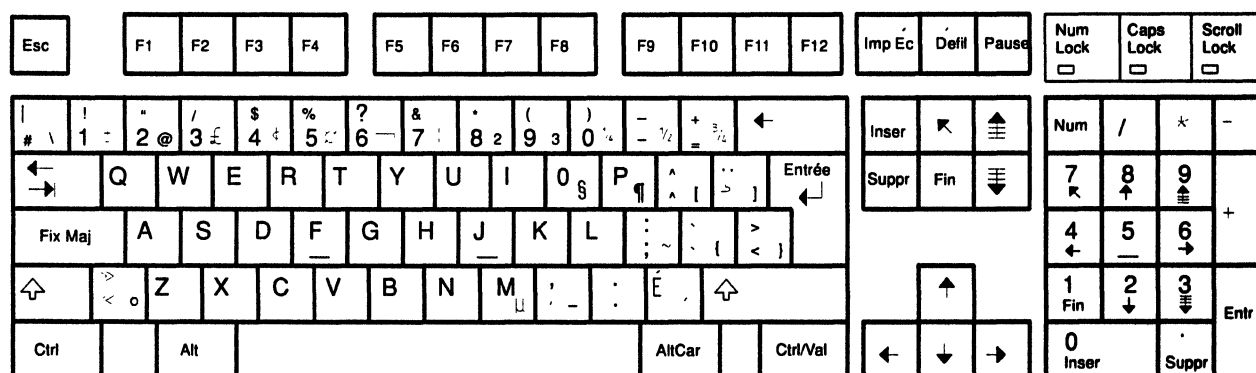


Figure 96. Canadian French

## Denmark

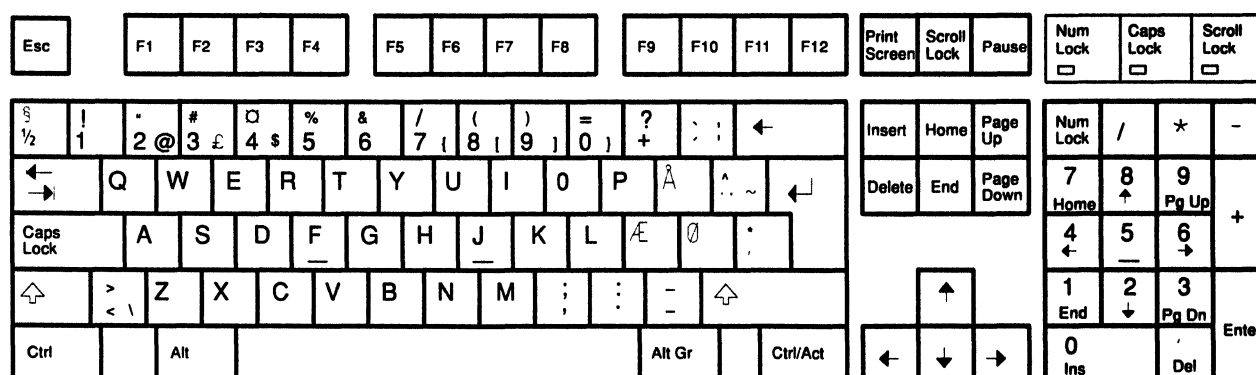


Figure 97. Danish



## France

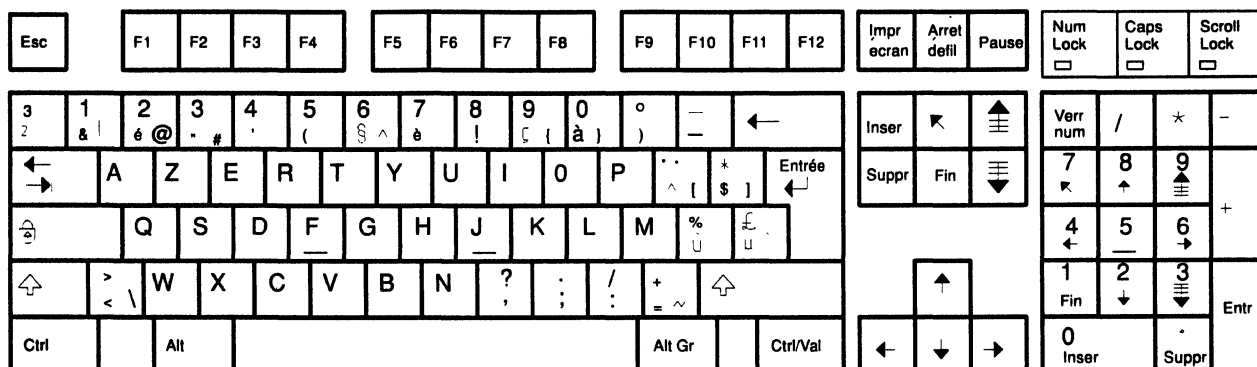


Figure 98. French

## German

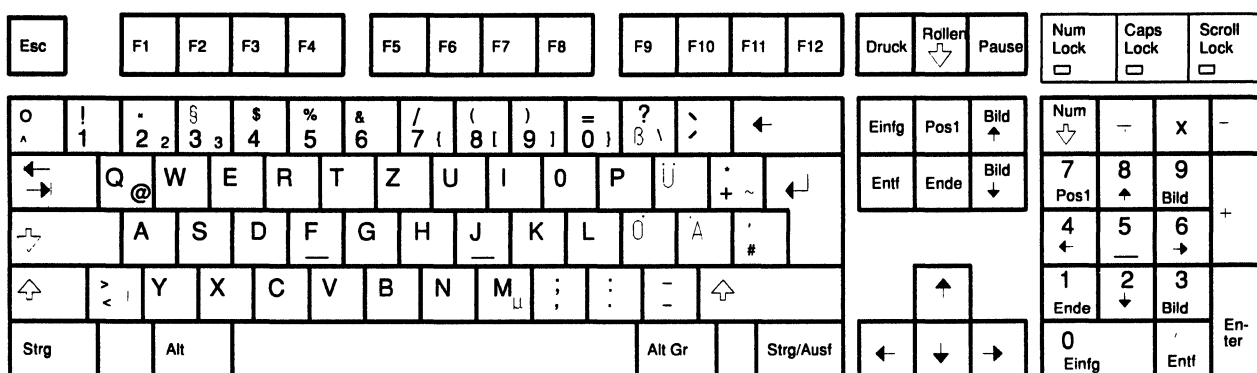


Figure 99. German

## Italy

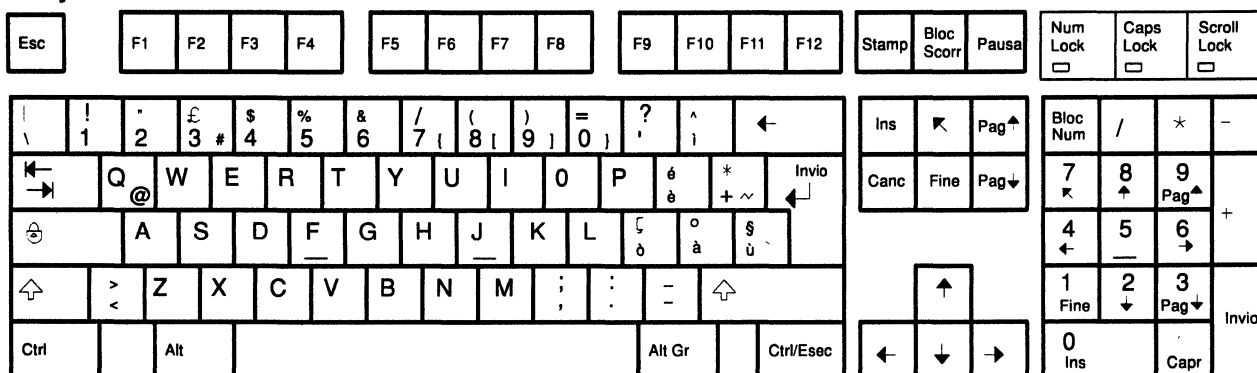


Figure 100. Italian



## Norway

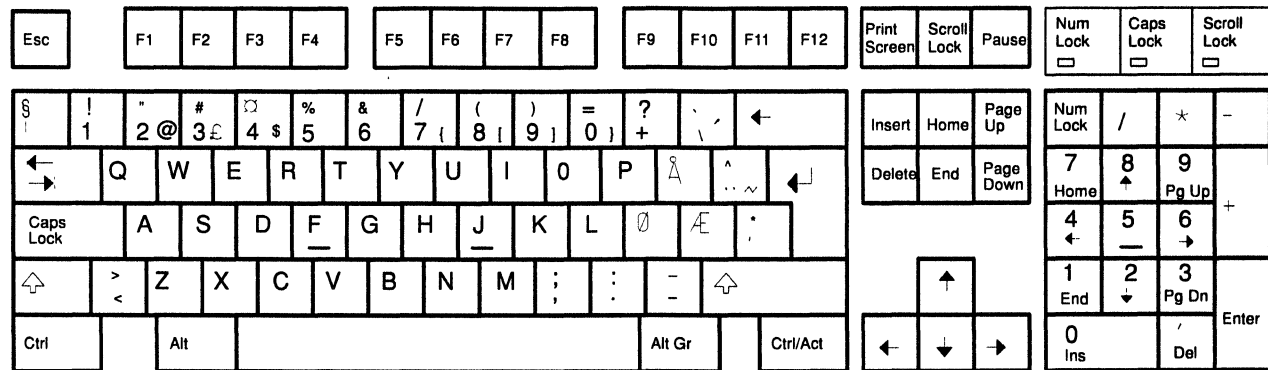


Figure 101. Norwegian

## Portugal

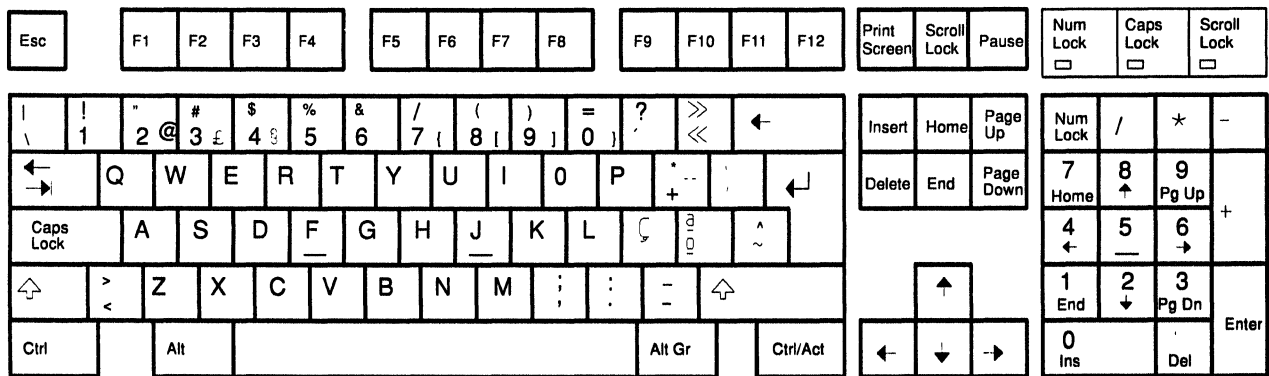


Figure 102. Portuguese

## Spain



Figure 103. Spanish



## Sweden/Finland

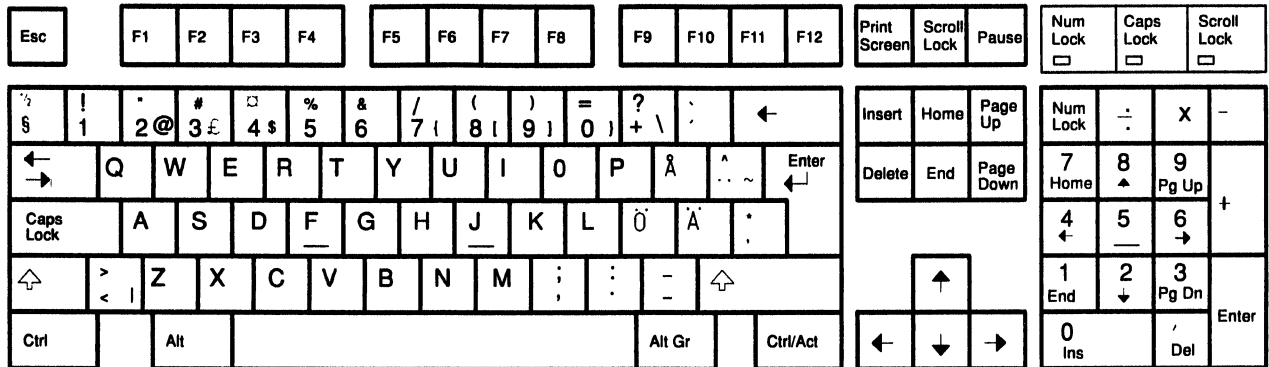


Figure 104. Swedish/Finland

## Swiss (Fr./Gr.)

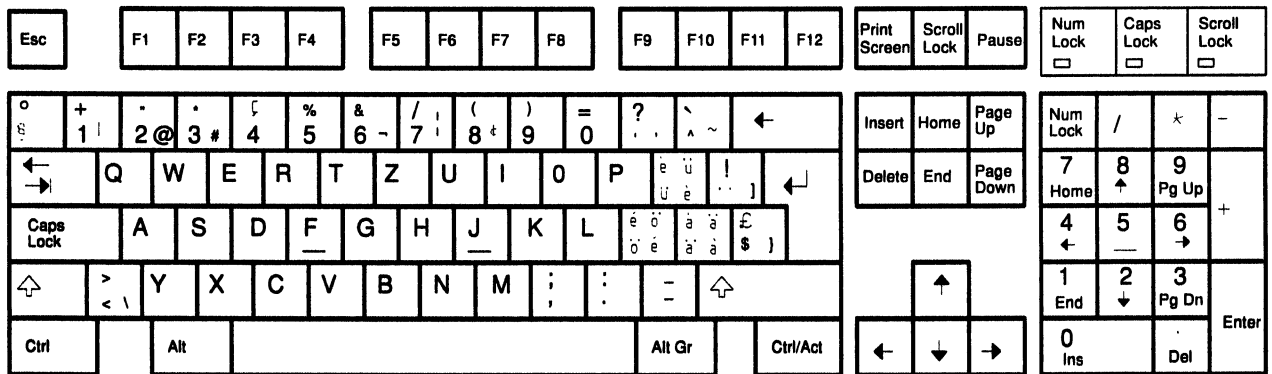


Figure 105. Swiss

## U.K.

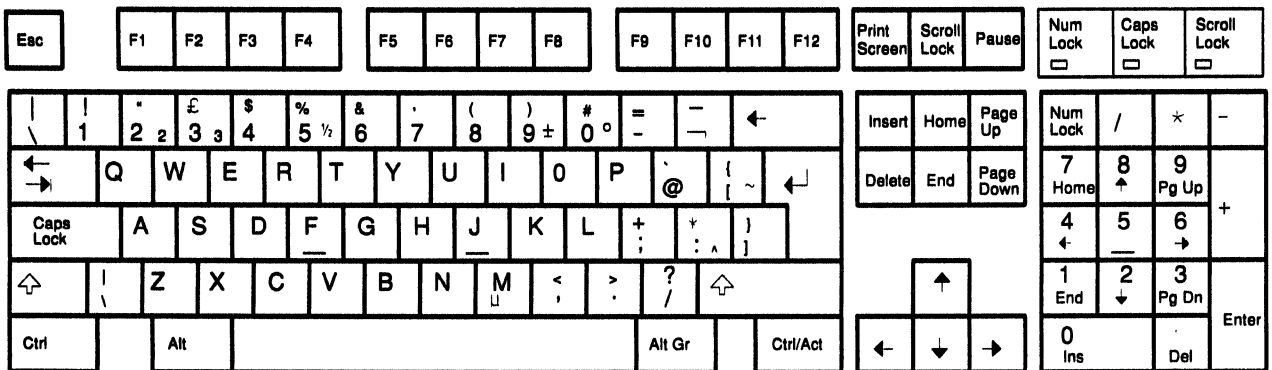


Figure 106. U.K. English



## United States

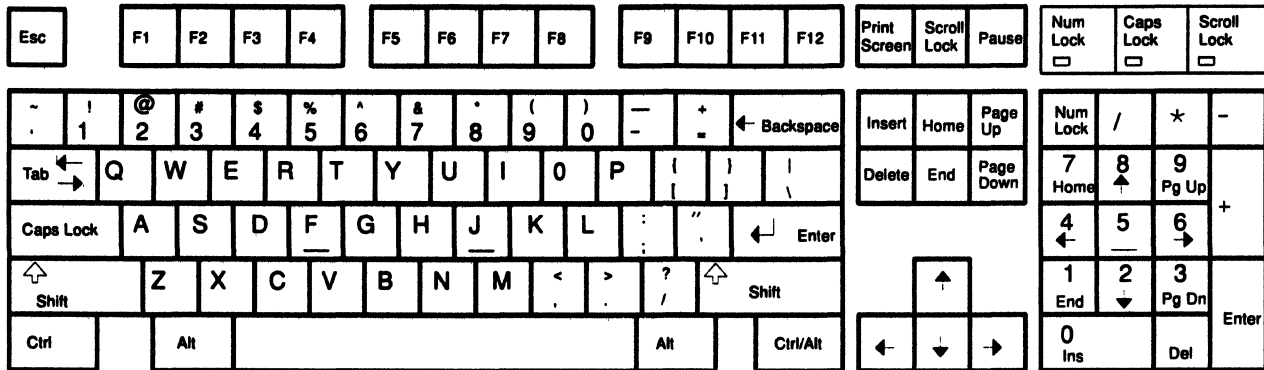


Figure 107. U.S. English



---

## Cables and Connectors

The keyboard has a detachable cable with a connector that attaches to a connector at the rear of the RISC System/6000 workstation. This shielded six-conductor cable provides power lines (+5 V dc) and two bidirectional signal lines. The cable is approximately 2.75 meters (9 feet) long. Figure 108 shows the pin configuration and signal assignments.

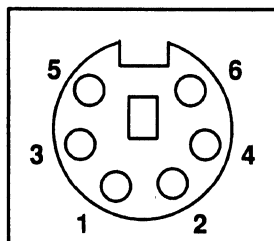


Figure 108. Keyboard Connector

Pin	Signal
1	Keyboard Data
2	Speaker Signal
3	Ground
4	+5 V dc
5	Keyboard Clock
6	Speaker Ground

---

## Specifications

Specifications for the keyboard are shown in the following sections.

### Power Requirements

The voltage and current shown as follows is required by the keyboard:

- +5 V dc  $\pm 10\%$
- 275 mA.

### Dimensions and Weight

Dimensions and weight of the keyboard are shown in the following list:

- Length: 492 mm (19.4 in.)
- Depth: 210 mm (8.3 in.)
- Height: 58 mm (2.3 in.), legs extended
- Weight: 2.25 Kg (5.0 lbs.).







---

# Chapter 9. 3-Button Mouse

## Chapter Contents

Introduction .....	9-3
Description .....	9-3
Operation Modes .....	9-3
Commands .....	9-4
Data Report .....	9-6
Error Handling .....	9-7
Data Frame .....	9-7
Data Transmission .....	9-7
Data Output .....	9-7
Data Input .....	9-8
Electrical Interface .....	9-8
Supply Voltage .....	9-8
Logic Voltage Levels .....	9-8
Operational Characteristics .....	9-9
Connector Specifications .....	9-9







---

## Description

This section contains mouse operational characteristics and modes, commands, and an explanation of data transmission between the mouse and the system. Connector specifications and voltage information for the mouse are also found in this section.

**Note:** For additional information refer to "Mouse Adapter" in the "Standard I/O" section of the particular RISC System/6000 technical reference manual.

## Description

The mouse is a cursor-positioning device that uses a rubber-coated ball and two mechanical encoders to indicate *X* and *Y* coordinates to the system to position the cursor. Three push-button switches are used to transmit select signals directly to the system. The mouse is connected to the system with a thin cable from a connector on the rear of the system unit. The ball is removable for cleaning.

---

## Operation Modes

The following describes the mouse modes of operation:

Mode	Description
<b>reset</b>	At startup or on receipt of a <b>reset</b> command, the mouse performs a self-test, transmits a completion code (X'AA') upon satisfactory completion, and an ID code (X'00') to the system. The following defaults are set: <ul style="list-style-type: none"><li>• Sampling rate equals 100 reports per second.</li><li>• Linear scaling.</li><li>• Stream mode.</li><li>• 150 counts per inch (6 counts per millimeter).</li><li>• 100 reports per second.</li><li>• Mouse is disabled.</li><li>• The mouse sends an error code of X'FC' followed by an ID code of X'00' immediately following an error to complete the diagnostics. The mouse is disabled and waits for additional commands from the system.</li></ul>
<b>stream</b>	In this mode, the mouse transmits a data report to the system if a button is pressed or released or if at least one count of movement has been detected since last reported. The rate of transfer is determined by the <b>set sampling rate</b> command and ranges from 10 samples per second to 200 samples per second. No transmissions occur while the mouse is motionless unless a button is operated, then the incremental movement report is 0.
<b>remote</b>	In this mode the mouse transmits data only in response to a <b>read data</b> command.
<b>wrap</b>	In this mode any byte of data sent by the system, except X'EC' or X'FF', is returned by the mouse.



---

## Commands

The following list contains descriptions of the mouse commands:

### Hex

Command	Description
FF	Reset: Causes the mouse to enter the reset mode and perform an internal self-test.
FE	Resend: This command is sent any time the mouse receives a command that is not valid.
F6	Set Default: This command reinitializes all conditions to the power-on default state.
F5	Disable: Used in stream mode to stop transmissions from the mouse. The mouse responds to all other commands while disabled. If the mouse is in stream mode, it must be disabled prior to sending it any command that requires a response.
F4	Enable: Allows the mouse to begin transmissions if it is in stream mode.
F3 AA	Set Sampling Rate: In stream mode the following sample rates are set to the value indicated by byte AA:

### Second

Byte AA	Sample Rate
X'0A'	10 per second
X'14'	20 per second
X'28'	40 per second
X'3C'	60 per second
X'50'	80 per second
X'64'	100 per second
X'C8'	200 per second.
F2	Read Device Type: Always receives a response of X'00'.
F0	Set Remote Mode: This command sets remote mode.
EE	Set Wrap Mode: Puts the mouse in wrap mode.
EC	Reset Wrap Mode: Resets wrap mode. The mouse returns to the previous mode of operation after receiving this command.
<b>Note:</b> If the mouse enters wrap mode while it is operating in stream mode, the <b>reset wrap mode</b> command causes the mouse to return to the stream mode in a disabled state.	
EB	Read Data: This command requests that all data defined in the data packet format be transmitted and is executed in either remote or stream mode. Following a <b>read data</b> command, the accumulators are cleared after the data transmission.
EA	Set Stream Mode: Sets stream mode.



**E9** Status Request: Causes the mouse to send the following 3-byte status report:

Byte	Bit	Configuration
1	7	Reserved
	6	0 equals stream mode, 1 equals remote mode
	5	0 equals disabled, 1 equals enabled
	4	0 = Scaling 1:1, 1 = Scaling 2:1
	3	Reserved
	2	1 equals left button pressed
	1	1 equals middle button pressed
	0	1 equals right button pressed
2	7-0	Current resolution setting (Bit 0 = LSB)
3	7-0	Current sampling rate (Bit 0 = LSB)

**E8 BB** Set Resolution: Sets the resolution to the following value specified by byte BB:

**Second**

Byte BB	3-Button
X'00'	25 per inch (1 per mm)
X'01'	75 per inch (3 per mm)
X'02'	150 per inch (6 per mm)
X'03'	320 per inch (12 per mm)

**E7** Set Scaling 2:1: Scaling is used to provide a course or fine tracking response. At the end of a sample interval in stream mode, the current X and Y data values are converted to new values. The sign bits are not involved in this conversion. The following shows the relationship between the input and output counts:

Input	Output
0	0
1	1
2	1
3	3
4	6
5	9
N(>=6)	2.0 x N

2:1 scaling is only performed in stream mode.

**E6** Reset Scaling: Restores 1:1 scaling.



---

## Data Report

When operating in stream mode, any change to the status bytes of the mouse device causes a data report to be sent at the end of a sample interval. Mouse movement of one count, or changing the button status from pressed position to released position, or vice versa, causes a change in the status bytes of the mouse device.

In the remote mode, a data report is sent in response to a **read data** command. The buttons are reported in their current state at the time of transmission.

The data report format shown in the following table is 3 bytes long and valid for both stream and remote modes.

Data Report Format		
Byte	Bits	Indication
1	0	Left button status, 1 equals pressed
	1	Right button status, 1 equals pressed
	2	Middle button status, 1 equals pressed
	3	Reserved
	4	X data sign bit (1 equals negative value)
	5	Y data sign bit (1 equals negative value)
	6	X data overflow (1 equals overflow)
	7	Y data overflow (1 equals overflow)
2	1–6	X data
	7	MSB of X data
	0	LSB of X data
3	1–6	Y data
	7	MSB of Y data
	0	LSB of Y data

The data values are in binary and the LSB indicates 0.1 inch (0.25 mm) of movement when operating with linear scaling at 10 counts per inch (four counts per mm) resolution. Negative values of X and Y data are expressed in twos complement where a 0.1 inch movement in the negative direction is expressed with all bits set to 1 and the Sign bit set to 1. After a transmission, the accumulators are set to 0.



---

## Error Handling

When the mouse receives any input that is not valid or any input with incorrect polarity, it issues a **resend** command (X'FE'). If two inputs that are not valid are received in succession, an error code of X'FC' is sent to the system.

Following a system transmission, the mouse responds within 25 milliseconds to commands that require a response, or if an error is detected in the transmission. If the mouse is in the stream mode, the system must disable the mouse before issuing any command requiring a response. When a command requiring a response is issued by the system, another command should not be issued until either the response is received or 25 milliseconds has elapsed. No more than four commands not requiring responses can be sent to the mouse in succession.

---

## Data Frame

The following describes the data frame:

Bit	Function
start bit	Always 0
0	Data (LSB)
1–6	Data
7	Data (MSB)
parity bit	Odd parity
stop bit	Always 1

---

## Data Transmission

During a data transmission, the 'clk' signal is used to clock serial data. The mouse generates the clocking signal when sending data to and receiving data from the system. The system requests that the mouse receive system data output by forcing the data signal line to inactive and allowing the 'clk' signal to go active.

Communication is bidirectional, using the clock and data signal lines. The signal for each of these lines comes from open collector devices, allowing either the mouse or the system to force a line to the inactive level. During a non-transmission state, the 'clk' and 'data' signals are both held at the (+) level.

## Data Output

When the mouse is ready to transmit data, it must first check for its own inhibit or system request-to-send status on the clock and data lines. If the 'clk' signal is low (inhibit status), data is continuously updated, and no transmissions are started. If the 'clk' signal is high and the data signal is low (request-to-send), data is updated. Data is received from the system and no transmissions are started by the mouse until the 'clk' and 'data' signals are both high.

If the 'clk' and 'data' signals are both high, the mouse proceeds to output a 0 Start bit, 8 Data bits, a Parity bit, and a Stop bit if a transmission is required. Data is valid prior to the falling edge of the 'clk' signal and beyond the rising edge of the 'clk' signal. During transmission, the mouse checks for line contention by checking for an inactive level on the 'clk' signal at intervals not to exceed 100  $\mu$ sec. Contention occurs when the system lowers the 'clk' signal



to inhibit the mouse output after a transmission has been started. If this occurs before the rising edge of the tenth clock (Parity bit), the mouse internally stores its data packet in its buffer and returns the 'data' and 'clk' signals to an active level. If the contention does not occur by the 10th clock, the transmission is complete.

Following a transmission, the system can inhibit the mouse by holding the 'clk' signal low until it can service the input or until the system receives a request from the mouse to send a response.

## Data Input

When the system is ready to send data, it first checks to see if the mouse is transmitting data. When the mouse is transmitting, the system can override the output by forcing the 'clk' signal to an inactive level prior to the tenth clock. When the mouse transmission occurs after the 10th clock, the system receives the data.

When the mouse is not transmitting or if the system chooses to override the output, the system forces the 'clk' signal to an inactive level for a period of not less than 100  $\mu$ sec while preparing for output. When the system is ready to output a 0 Start bit (data line low), it allows the 'clk' signal to go to a (+) level. This state is checked by the mouse at an interval not to exceed 10 milliseconds.

When a request-to-send condition is detected, the mouse clocks in 11 bits of data. Following the tenth clock, the mouse checks for a (+) level on the data line, and if found, the mouse forces the data low and clocks once more. This signals the system to return to the ready state so it can accept input or enter the inhibit mode until ready. If the data signal is detected at an inactive level following clock 10, a framing error has occurred and the mouse continues clocking until the data signal is high. The mouse then clocks the Line Control bit and requests a **resend**.

For each system command or data transmission that requires a response, the system waits for the mouse to respond before sending its next output. The response must be within 20 ms, unless the system inhibits the mouse output or inhibits the data transmission from the system that requires a response.

When a system command is initiated, the data transmission and the associated response is not valid. If there is a parity error, the system retransmits the command or data. If the response is still not valid or has a parity error after two retries, the system resets the mouse.

---

## Electrical Interface

The RISC System/6000 Standard I/O supplies the electrical interface to the mouse, providing supply and logic level voltages.

### Supply Voltage

One voltage level is available to the mouse at the system connector.

$V_{cc} = 5\text{ V dc} \pm 10\%$

No more than 100 mV peak-to-peak differential noise and ripple is present on the +5 volt line.

On startup and shutdown the mouse can tolerate a voltage overshoot of  $V_{cc} + 30\%$  or a voltage undershoot of  $V_{cc} - 0.3\text{ V}$  with no damage.

### Logic Voltage Levels

The term – level is defined to be equivalent to a voltage,  $V$ , measured between a signal source and a network ground such that  $0.0\text{ V} \leq V \leq +0.7\text{ V}$ . A signal at a – level is a logical 0.



The term + level is defined to be equivalent to a voltage, V, measured between a signal source and a network ground such that  $2.4\text{ V} \leq V \leq +5.5\text{ V}$ . A signal at a + level is a logical 1.

---

## Operational Characteristics

Operational characteristics for the mouse are described in the following list:

Characteristic	Description
resolution	Programmable to 25, 75, 150, or 320 counts per inch. Default equals 150.
sampling rate	Programmable to 10, 20, 30, 40, 60, 80, 100, or 200 reports per second. Default equals 100.
data modes	Stream (default), remote, or wrap.
scaling	1:1 or 2:1.
power	+ 5 V dc, $\pm 10\%$ , < 125 mA.
protocol	Clock and Data Interface.

---

## Connector Specifications

Figure 109 shows the mouse connector.

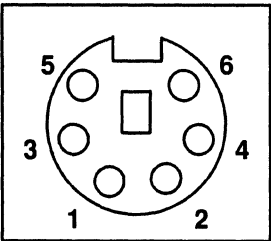


Figure 109. Mouse Cable Connector

The following list contains mouse connector pin numbers and signals:

Pin	Signal
1	Mouse Data
2	Reserved
3	Ground
4	+5V dc
5	Mouse Clock
6	Reserved







---

## Chapter 10. Micro Channel Adapter Support

### Chapter Contents

Description .....	10-3
IBM Micro Channel Optional Features Supported .....	10-4
Configuration .....	10-5
RISC System/6000 Configuration Procedures .....	10-6
Devices Configuration .....	10-6
ADF's .....	10-6
Device Drivers .....	10-6
Other Micro Channel Adapter Design Considerations .....	10-6
Adapter Configurations Supported .....	10-7
Dimensions .....	10-7
Raw Card Thickness .....	10-7
Thermal .....	10-7
Assembled Card Thickness .....	10-8
Card Layout With Metal Bracket .....	10-8
Top Card Connectors .....	10-8
Power .....	10-9
Recommended Maximum Current Per Adapter .....	10-9
Micro Channel Architecture Deviations .....	10-10







---

## Description

This chapter provides some basic information specific to the RISC System/6000 and the support of Micro Channel adapters. Topics included in this chapter are as follows:

- The first section provides a description of the Micro Channel optional features which will be supported in the RISC System/6000 family.
- The second section provides a brief overview of the RISC System/6000 configuration procedure describing how Micro Channel adapters are configured in the system.
- The third section, titled "RISC System /6000 – Other Micro Channel Adapter Design Considerations" complements the *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture* document. That document includes 2 chapters. The first chapter is "Micro Channel Architecture". The second chapter titled "Micro Channel Adapter Design", provides some basic guidelines to design adapters for the Micro Channel architecture 16-bit and 32-bit products. This third section provides additional information not included in the referenced document, such as physical specifications for a larger card type, and a current limitation specification for Micro Channel adapters installed in the RISC System/6000 family.
- The last section of this chapter lists deviations from the Micro Channel architecture.



---

## IBM Micro Channel Optional Features Supported

There are a number of features that are optional in the IBM Micro Channel Architecture that are supported in the RISC System/6000 family. The optional features that are supported in the RISC System/6000 family are:

- **Streaming Data Operations:** The streaming data procedure provides a faster data transfer rate than the basic transfer procedure. The RISC System/6000 family provides for 16 and 32-bit Bus Master Streaming Data with the IOCC operating as the slave.
- **Address and Data Parity:** The address and data parity support (odd parity) provides the ability to detect and recover from parity errors on the Micro Channel.
- **Video Extension:** 16-bit and 32-bit adapters with video extension may be installed in the RISC System/6000 system. However the video extension function is not supported. The appropriate signals for the video extension function are provided pull-up resistors but the signals are not bussed. The Micro Channel 'oscillator' (OSC) signal is also provided for the video extension. The video extension connector pins are supported as follows:

A Side	Pin	B Side
Pullup-2K	10	Pullup-2K
Pullup-2K	9	Pullup-2K
Pullup-2K	8	Pullup-2K
Ground	7	Pullup-2k
Pullup-2K	6	Pullup-2K
Pullup-2K	5	Pullup-2K
OSC (14.318 MHz)	4	Pullup-2K
Ground	3	Pullup-2K
Pullup-2K	2	Pullup-2K
Pullup-2K	1	Pullup-2K



- **Matched Memory:** 32-bit adapters with matched-memory extension may be installed in the RISC System/6000 system units. However the matched memory function is not supported. The appropriate signals for the matched memory function are bussed and pull-up resistors are provided. The matched memory connector pins are supported as follows:

A Side	Pin	B Side
Pullup-2.5K	1	Pullup-2.5K
Ground	2	Pullup-2.5K
Pullup-2.5K	3	Pullup-2.5K (Reserved)
Pullup-2.5K (Reserved)	4	Ground

- **Asynchronous Channel Check (–CHCK):** To maximize the system and Micro Channel availability, use of the asynchronous –CHCK should be avoided in the RISC System/6000 system. If the asynchronous –CHCK is used by I/O adapters, the asynchronous –CHCK is detected and all Micro Channel I/O activity is momentarily suspended, and this usually results in a system reset.

---

## Configuration

When a Micro Channel adapter is plugged into a system, the adapter must be made known and usable within that system. The RISC System/6000 system and PS/2 systems accomplish this by using two different procedures. The procedure is called "SETUP" in a PS/2, and is called "Devices Configuration" in a RISC System/6000 system. In "System Configuration" in the Micro Channel Architecture chapter, it explains that the system master configures a system using setup cycles. The system configuration procedure supports the identification of the adapters that reside within the system (by reading the adapter ID's in POS) and the configuration of those adapters (by writing configuration data into POS). Programmable Option Select (POS) registers are a set of software programmable registers located on the adapter accessible during setup cycles. The Option Diskette supplied with an existing Micro Channel adapter contains the specific adapter configuration data in a software module called the Adapter Description File (ADF). In the RISC System/6000 family, the appropriate ADF information is "prepopulated" on the fixed disk in the "Predefined" database for the adapters initially supported (Predefined is a set of object classes for the predefined configuration data). For a description of the "SETUP" procedures for the PS/2 systems, see the "IBM Personal System/2 Hardware Interface Technical Reference".



---

## RISC System/6000 Configuration Procedures

The following sections give a very brief overview of the RISC System/6000 configuration procedures (Including the Micro Channel I/O adapters), and reference the appropriate chapters in the RISC System/6000 AIX Version 3 software documentation for details.

### Devices Configuration

"Devices Configuration" is a procedure which performs a series of steps to make a device known and usable in the RISC System/6000 system. Devices Configuration includes a number of software modules called Device Methods. Devices Configuration is performed during both system IPL and runtime. Specific "Predefined" information must be provided for each adapter. In the RISC System/6000 family, all adapter predefined information is prepopulated in the predefined database on the fixed disk for all adapters initially supported. Device Methods use the devices predefined information to resolve all device configuration options and creates the "Customized Database" (which is the systems "Configured" database). There are five device methods that are needed to support a device. They are the define, configure, change, undefine, and unconfigure methods. For detailed information on the Device Methods, see "Device Methods" in *"Programming Device Support for AIX Configure subsystems"*. The appropriate adapter configuration data is also written (By Device Drivers) in each adapters set of POS registers. When adding a device to the system, there are other areas that must be considered in addition to writing device methods. These areas include how the data base, device SMIT dialogues, device drivers, ADF utility, installp, and diagnostics, all relate to your device. "Device Methods" in *"Programming Device Support for AIX Configure subsystems"* defines and describes all these items in detail or references the appropriate documentation.

### ADF's

A utility called **adfutil** command provides the capability to field merge Micro Channel adapter resource information (ADF's) for Micro Channel adapters into the predefined information in the AIX Version 3 Configuration Database (See **adfutil** command section in "How To Write Device Methods" for more detail). This is accomplished by processing information (ADF's) found on DOS formatted diskettes provided with the Micro Channel adapter hardware. A new device driver for an existing Micro Channel adapter must also be written specifically for the RISC System/6000 system before it can be installed.

### Device Drivers

For information on writing an adapter device driver see the chapter "How to Write Device Drivers" in the software documentation.

---

## Other Micro Channel Adapter Design Considerations

This section complements another RISC System /6000 document, *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture*. That document includes 2 chapters. The first chapter is "Micro Channel Architecture". The second chapter titled "Micro Channel Adapter Design", provides some basic guidelines to design adapters for the Micro Channel architecture 16-bit and 32-bit products. This third section will include additional information not included in the referenced document such as physical specifications for a larger card type, and a current limitation specification for Micro Channel adapters to be installed in the RISC System/6000 system.



---

## Adapter Configurations Supported

The I/O Board provides channel connectors to support the following adapter configurations.

- 16-bit adapter
- 32-bit adapter
- 16-bit adapter with video extension
- 32-bit adapter with video extension
- 32-bit adapter with match memory

The physical specification for the 32-bit adapter with video extension is included in this section along with the physical specifications for the Type 5 card described below. Physical specifications for the other adapter configurations are described in *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture*.

---

## Dimensions

RISC System/6000 packaging physically supports two card form factors, Type 3 and Type 5.

- **Type 3 Cards:** The Type 3 nominal raw card physical size is 3.475 x 11.5 inches. For details of the Type 3 card physical specifications, see the "Micro Channel Adapter Design" chapter in *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture* (The Type 3 card is also known as the Standard raw card size in that document). The raw card is 11.5 inches in length and the retainer that fastens to the end of the card extends the overall length to approximately 12.2 inches. The retainer sits in an adapter channel that will support the card and prevent it from tilting sideways and coming into contact with adjacent cards. In some cases the card is retained on its top edge (the 3.475 height). Card retainers are required on this card.
- **Type 5 Cards:** The Type 5 nominal raw card physical size is 4.825 x 13.1 inches. For details of the Type 5 card physical specifications, see the figures at the end of this section. The raw card is 13.1 inches in length and sits in a channel that will support the card and prevent it from tilting sideways. In some cases the card is retained on its top edge (the 4.825 height). No card retainer is allowed on this card.

Cards (Type 3 and Type 5) that are shorter than the dimensions indicated above will require a special card retainer which extends the card length to allow proper retention once installed.

## Raw Card Thickness

The bus connector is designed for a feature card thickness of 0.062 ±.005 inch.

## Thermal

The adapter design should avoid clustering of high temperature components. No component should exceed its maximum thermal rating. A figure is shown at the end of this section illustrating areas on each card Type where placement of components that tend to operate warm should be avoided.



## Assembled Card Thickness

For minimum feature card spacing and maximum component height specifications, see "Micro Channel Adapter Physical Specifications" in the *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference — Micro Channel Architecture* document. Deviations from the thicknesses specified can result in touching of adjacent cards, affect cooling due to reduced air flow, and may limit placement of I/O cards within the machine.

## Card Layout With Metal Bracket

The metal bracket covers 0.200 inch of the card when installed. When laying out the card, do not install any components near the I/O end of the card that would interfere with the installation of the bracket. During the wiring operation do not route lands under the bracket because the bracket is located just above the circuit board. If the bracket was slightly bent, the bracket could short to the lands. Also use care when placing components near the bracket whose leads could short to the bracket. The metal bracket location is the same for both Type 3 and Type 5 cards.

## Top Card Connectors

There are specific requirements for card designs that use a top card connector (See the drawings at the end of this section for details). The raw card drawings indicate a specific area that must be used for the top card connector. The top of the connector must not extend beyond 0.276 inch above the top edge of the Type 3 card. For the Type 5 card, the top edge connector must be recessed and not extended above the top edge of the card. Above these points, the connector may interfere with the covers. If the connector is located at any point other than that specified, the cabling will impact airflow through the system. Note from the drawings that Type 3 cards designed specifically for the RISC System/6000 system may have a top card connector position other than that specified for the PS/2. Type 3 card designs intended for both RISC System/6000 and PS/2 systems which use top card connectors, should use the top card connector position specified for use in the PS/2. See the appropriate figures in the "Micro Channel Adapter Design" chapter.



## Power

The following figure illustrates the recommended maximum currents for adapters installed in the RISC System/6000 system. This allows for maximum slot configurability in each RISC System/6000 system. Figure 110 also provides the absolute maximum currents that may be allowed per individual card slot and for all slots. Care must be taken when designing cards which exceed the recommended maximum current specifications, as this results in system configuration limitations (such as not populating some card slots or placement of lower power cards or empty card slots next to high power adapters). The RISC System/6000 current capabilities exceed those defined in the Micro Channel architecture. To enable portability across all Micro Channel systems, the Micro Channel architecture current requirements should be met.

		RISC System/6000 Models			
		320	520,530, 540, 730	930	
RECOMMENDED maximum current per adapter (amps)	+5 Volts +12 Volts -12 Volts	3.12 0.25 0.10			Note 1
Absolute maximum current per adapter slot (amps)	+5 Volts +12 Volts -12 Volts	6.5 1.0 0.4	6.5 1.0 0.5	6.5 1.0 0.4	Note 2
Maximum current allowed for all card slots combined (amps)	+5 Volts +12 Volts -12 Volts	19.0 2.0 0.4	25.0 2.0 1.9	25.0 2.7 2.0	Note 3

Figure 110. RISC System /6000 Channel Load Currents

### Notes:

1. The RECOMMENDED maximum current ratings allow for maximum configurability of all Micro Channel card slots in each RISC System /6000.
2. The total current for all Micro Channel cards installed in any particular RISC System/6000 system must not exceed the maximum total rated currents listed.
3. The RISC System/6000 system can provide the absolute maximum currents listed per Micro Channel card slot. However, cards designed to exceed the RECOMMENDED maximum current limits per adapter, result in system configuration limitations such as fewer cards allowed per system, etc. Adapter cards with power dissipation of 20 watts or more should be separated by empty card positions or lower power adapters when possible.



---

## Micro Channel Architecture Deviations

The following are implementation specific RISC System/6000 deviations from the Micro Channel architecture. It has been verified that the supported set of adapters function satisfactorily on the Micro Channel in the RISC System/6000 systems with the stated deviations.

- When the RISC System/6000 I/O Channel Converter (IOCC) is an I/O slave on the Micro Channel, it will hold the '–CHRDYRTN' line (in some situations) longer than the specified 3.5 microseconds ('–CHRDYRTN' may be held inactive 3.8 microseconds.)
- The IOCC does not meet the timing specification T31 for the BE(0–3) lines. BE(0–3) becomes valid with the falling edge of '–ADL' instead of the specified time. This still provides sufficient time for the 'BE(0–3)' signals to be valid on the bus to meet the T33 timing parameter for slaves.
- The RISC System/6000 I/O Channel Converter (IOCC) as a slave does not check data parity on 2–byte transfers from 16–bit bus masters (such as the Token Ring High-Performance Adapter and the 4-Port Multi Protocol Adapter) when the lower 2–bits of the address are binary '10'.
- The IOCC activates '–SD STROBE' during Programmed I/O (PIO) operations, though streaming data operations with the IOCC as the master are not supported.
- The IOCC does not meet the T1 timing specification during REFRESH cycles. Address is valid coincident with Status active instead of 10 ns prior to Status active.
- The RISC System/6000 Standard I/O does not activate 'Channel Check' (–CHCK) for address parity errors. If an address parity error occurs, the Standard I/O will not be selected and will not activate '–CD SFDBK'. The lack of '–CD SFDBK' will be detected by the IOCC and reported as an exception.
- The RISC System/6000 Standard I/O does not have a parity enable/disable POS bit. In the RISC System/6000, the parity function is enabled.



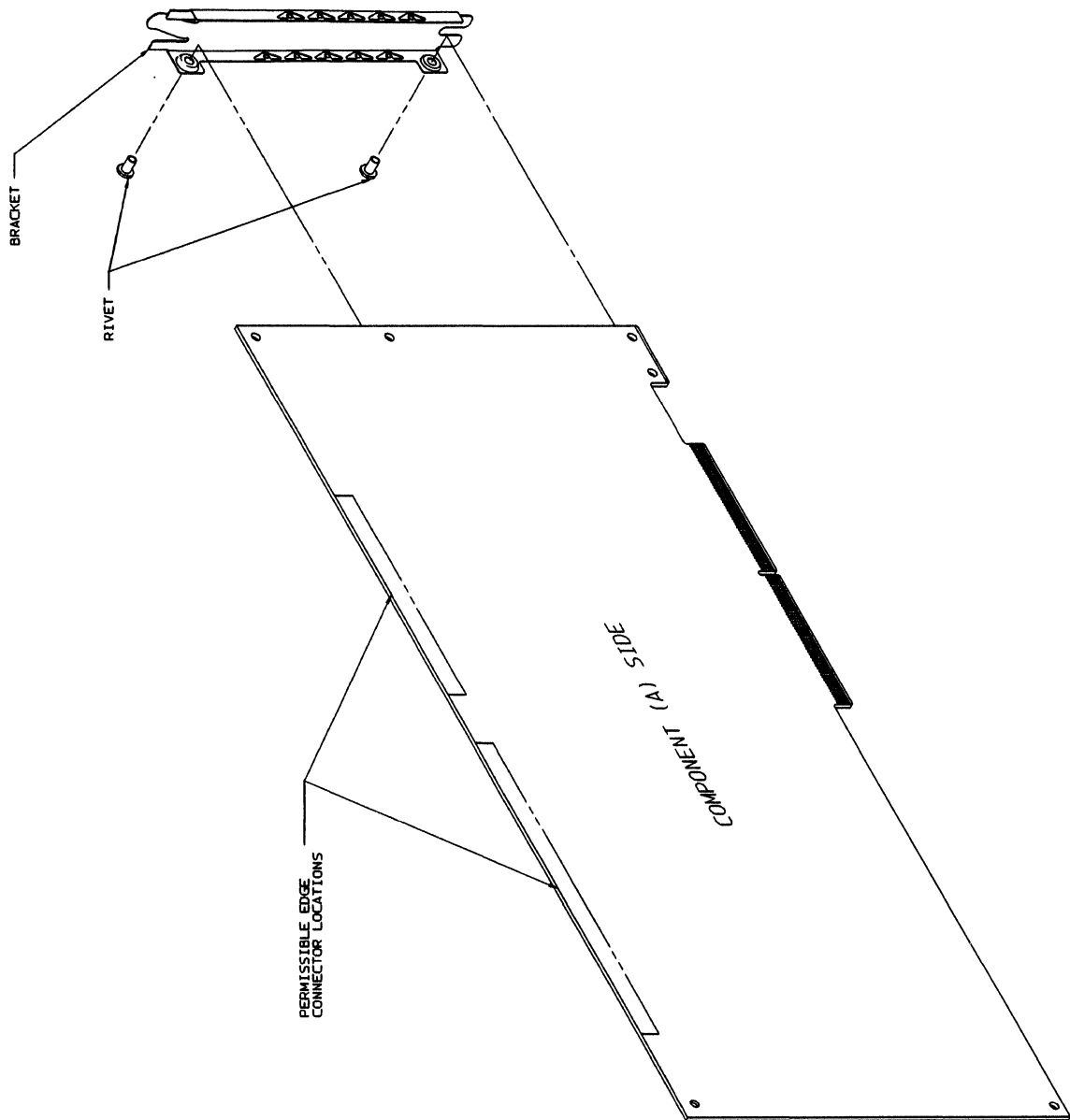


Figure 111. Type 5 Adapter Assembly







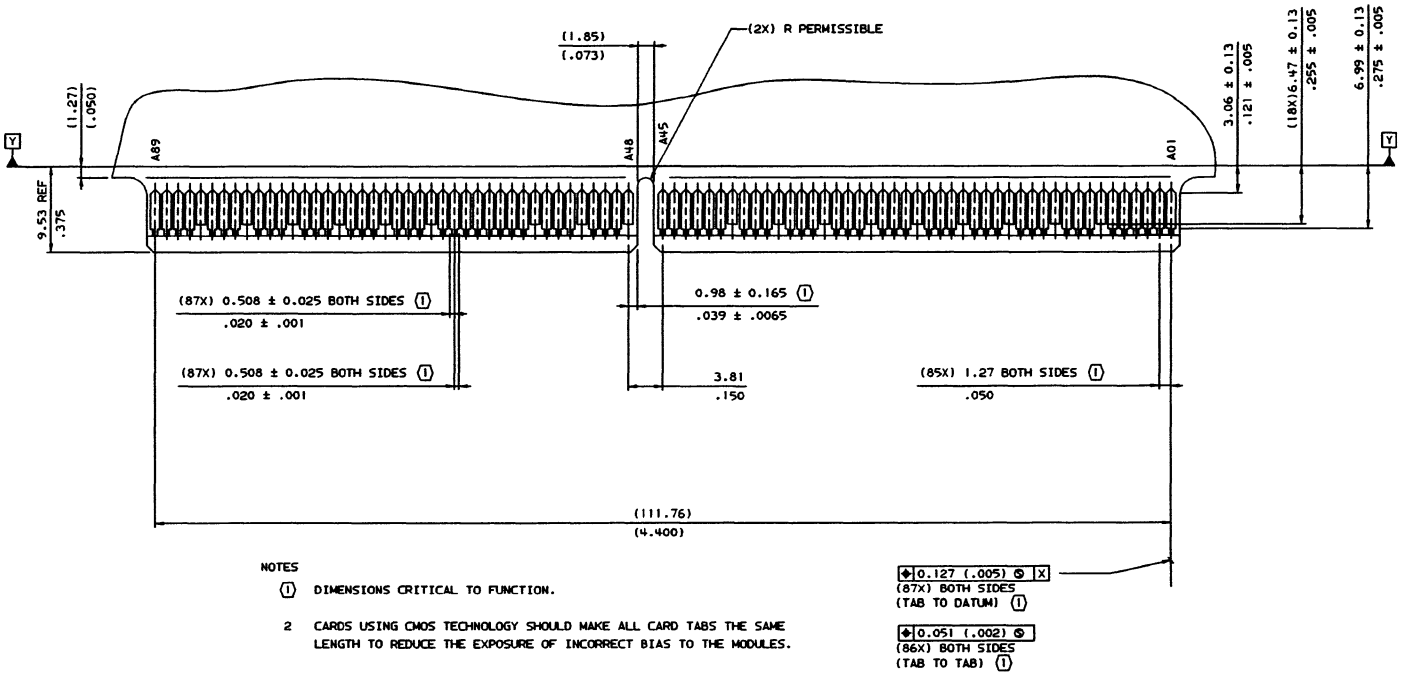


Figure 113. Connector Dimensions (32-Bit)



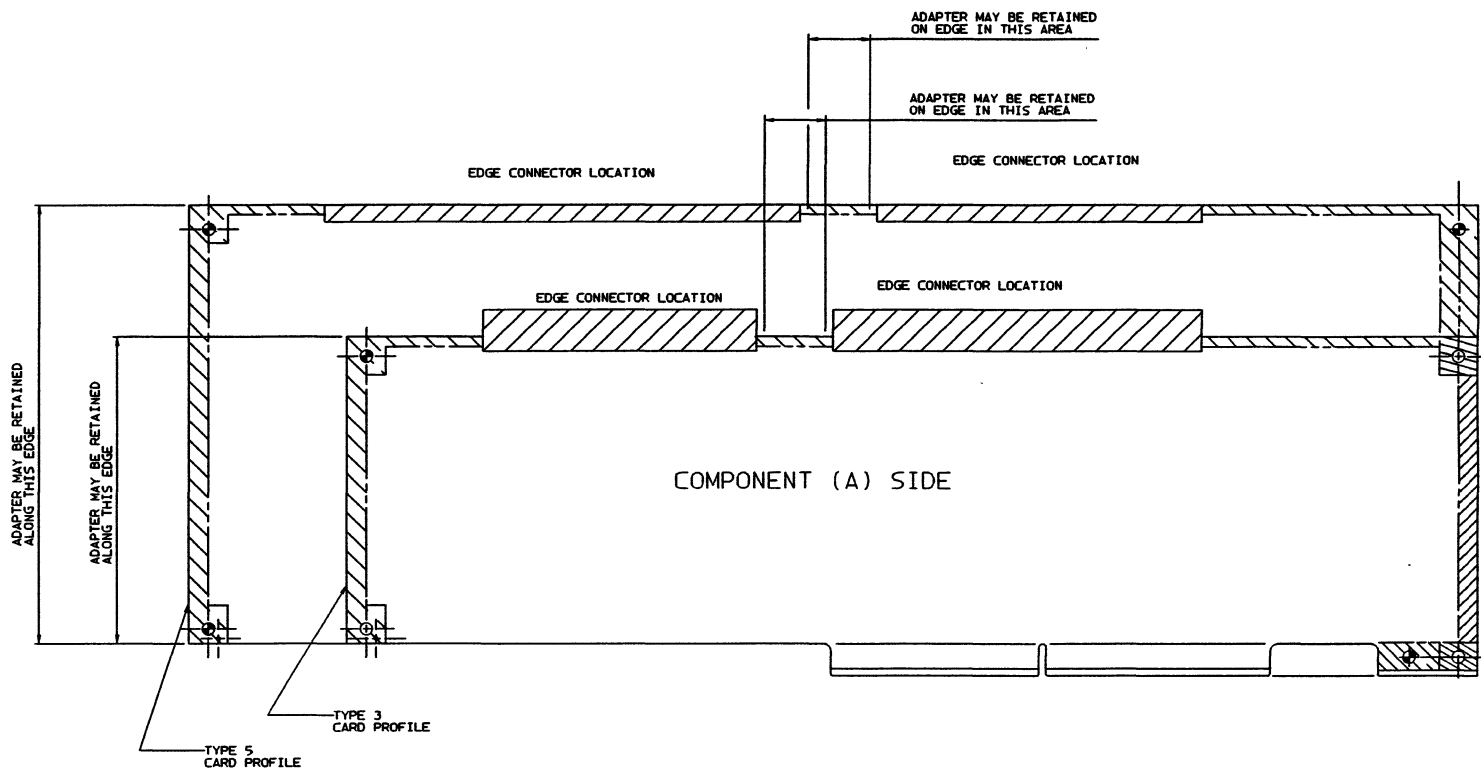








Figure 116. Type 5 and Type 3 Card Profiles





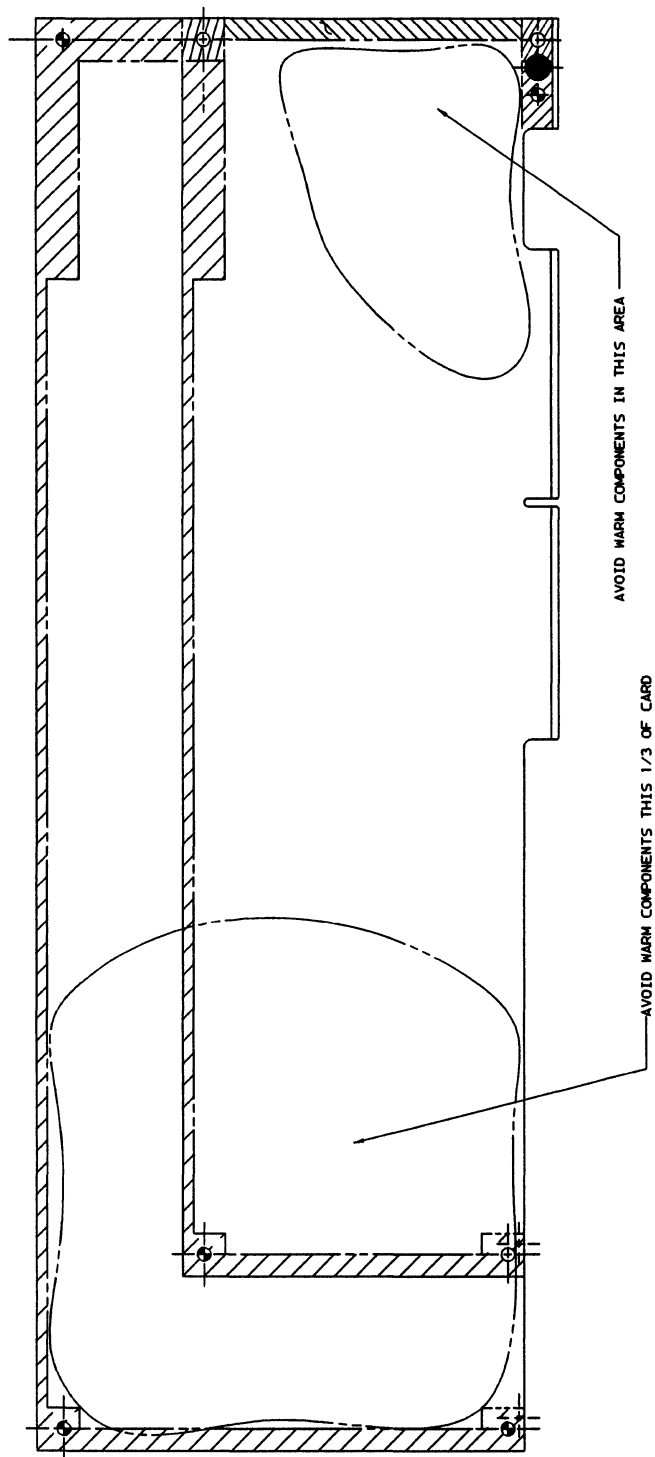
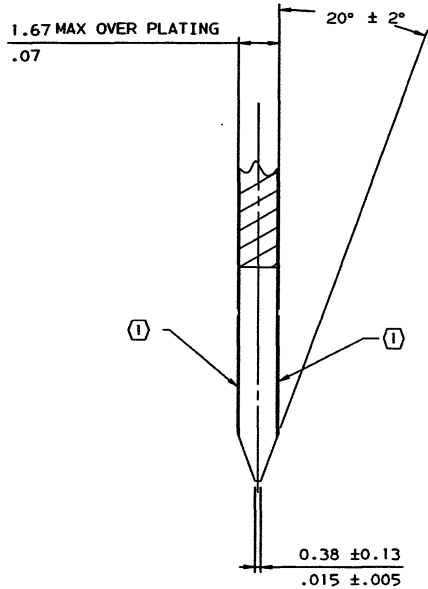


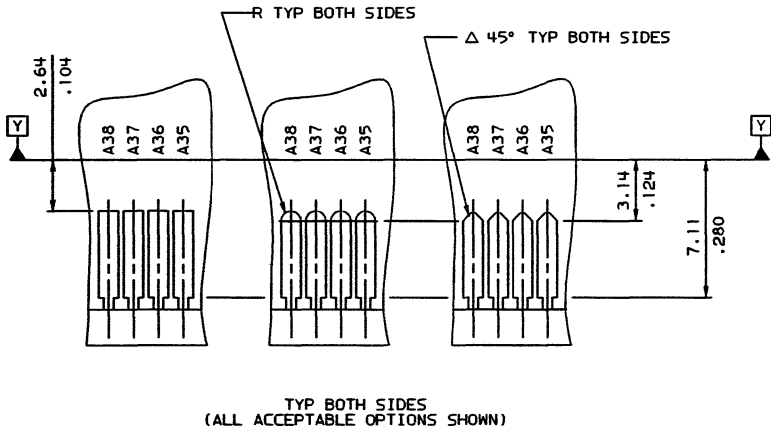
Figure 117. Component Locations for Thermal



Figure 118. Connector (Common Detail)



- NOTES
- ① ALL TABS, BOTH SIDES, TO BE GOLD PLATED 0.0018 mm (.00007 in) MIN OVER 0.00254 mm (.0001 in) MIN NICKEL AND CHAMFERED AS SHOWN.



























© IBM Corp. 1990

International Business Machines  
Corporation  
11400 Burnet Road  
Austin, Texas 78758-3493

Printed in the  
United States of America  
All Rights Reserved

SA23-2643-00

SA23-2643-00

