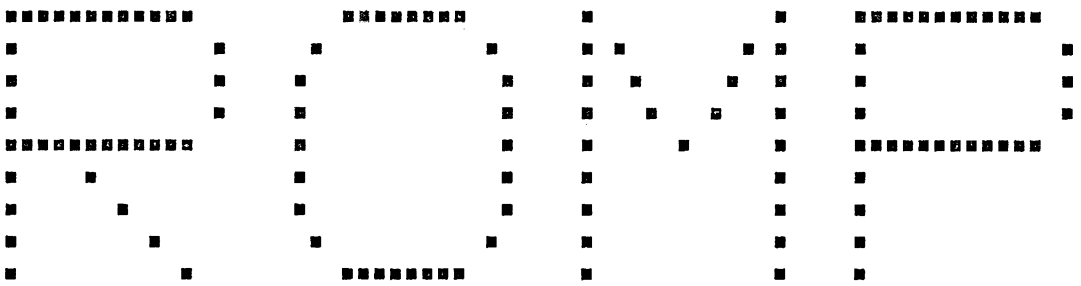
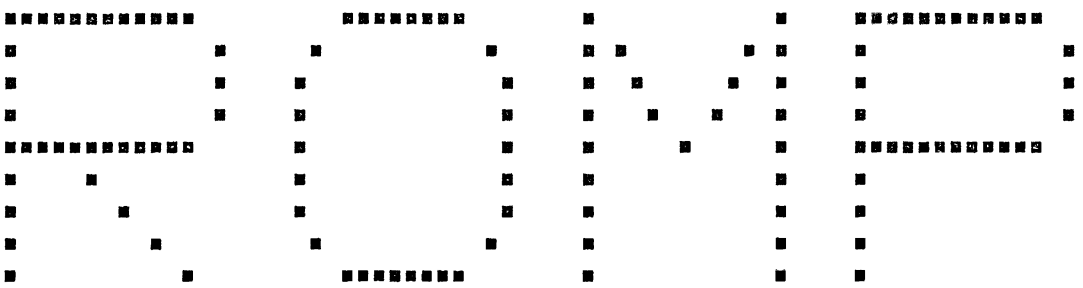
IBM CONFIDENTIAL-RESTRICTED DO NOT COPY



FUNCTIONAL SPECIFICATION
December 1, 1983
E.C.# A07313
P.N. 6080439

IBM CONFIDENTIAL-RESTRICTED DO NOT COPY



FUNCTIONAL SPECIFICATION
December 1, 1983
E.C.# A07313
P.N. 6080439

IBM Confidential Restricted. DO NOT COPY

ROMP Functional Specification
December 1, 1983

Tom Whiteside
F61/045
Austin, Tx
T.L. 678-9791

Document Printed July 25, 1984

COPY _____

ROMP Functional Specification

Document Number 6080439

December 1, 1983

Tom Whiteside

IBM Entry Systems Division
Dept. F61 Bldg. 045
11400 Burnet Road
Austin, TX 78758
Tie Line 678-9232

COPY _____

ROMP Functional Specification

Document Number 6080439

December 1, 1983

Tom Whiteside

IBM Entry Systems Division
Dept. F61 Bldg. 045
11400 Burnet Road
Austin, TX 78758
Tie Line 678-9232

CONTENTS

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

1.1 DOCUMENT OVERVIEW

This document is the functional specification for the ROMP processor. Information pertaining to the ROMP processor's organization, its instruction set, its I/O capabilities, and its RAS facilities are contained in this document. Other documents are listed in the Appendix which provide additional detailed hardware and software information.

1.2 ROMP OBJECTIVES

1. Provide an architected address space of 32-bits. (Choice of 24-bit or 32-bit addressing mode).

2. Provide high performance with fast or slow storage (three MIPS typical with 200 nsec storage).

3. Provide the capability for dynamic address translation.

4. Provide system integrity through the use of storage protect and problem/supervisor states.

5. Provide an efficient target for the PL.8 Compiler.

6. Low power dissipation.

7. Improved debug facilities for IBM products.

1.3 ROMP PROCESSOR HIGHLIGHTS

The ROMP processor architecture provides comprehensive facilities for support of many different IBM products. The highlights of the ROMP processor are described here.

The ROMP processor provides 32-bit storage addresses which permits up to 4.3 gigabytes of main storage to be directly accessed. Both instructions and data are contained in main storage.

The address of data in main storage is computed from two values, a base and a displacement, at the time the data is accessed. When data is arranged in blocks, a single base register permits accessing the entire block. Base/displacement addressing allows address field abbreviation in instructions. A base register

permits the instructions which access the data in main storage to be independent of the location of the data.

The processor provides sixteen 32-bit general purpose registers, which are not part of main storage. All arithmetic and logical functions are performed on the general purpose registers which may also function as base registers for base/displacement addressing. The only data operations provided on main storage are loading of data from main storage into a general purpose register and storing of data into main storage from a general purpose register.

The processor also provides sixteen 32-bit system control registers. The system control registers contain the current status of the system. All system control registers may be inspected by the program, and several also can be explicitly modified by the program.

The instruction set includes instructions for accessing storage, arithmetic and logical computations, program and system control including branching, and input/output. An instruction is either two or four bytes in length. The instruction set has been tailored for performance, storage efficiency and function, and has been demonstrated to give high performance in spite of its simplicity.

The ROMP processor supports dynamic address translation in the storage controller. This feature allows a large virtual address space to be mapped into smaller physical address space. ROMP provides an exact interrupt mechanism that allows the processor state to be saved when a storage exception condition occurs. The processor state can then be reloaded after the exception has been handled, and execution of the program continued.

ROMP provides a supervisor state in which all instructions are valid, and a problem state in which only instructions that cannot be used to affect system integrity are valid. ROMP also provides a storage protect mechanism that allows address checking to be implemented in the storage controller. These two features insure that system integrity can be maintained at all times.

The ROMP processor provides a priority interrupt structure. An interrupt at a high priority level preempts ongoing activity at a lower priority level. When an interrupt occurs, only the basic processor status is changed. This primitive status switching permits the flexible programming of dispatching mechanisms.

Two levels of input/output support are provided. For low performance data transfer, programmed I/O (PIO) is synchronous to the issuing program. For high performance data transfer, direct memory access (DMA) permits transferring of data asynchronously to the program.

The ROMP processor implements a high-performance, 32-bit storage channel called the ROMP Storage Channel (RSC). The RSC supports

the high data rate needed for instruction execution, and can also be used for high-speed I/O devices.

The ROMP processor provides facilities for logging errors in the system. The logged information can be used as a basis for isolation of a failing component.

1.4 PROGRAMMING SUPPORT OVERVIEW

The ROMP processor was designed to be an efficient target for the PL.8 high-level language. PL.8 is a derivative of PL/I and has been demonstrated to be well suited to systems programming needs. The compiler for PL.8 uses program flow optimization techniques which produce efficient code in terms of both storage and performance. The high-level language provides improved programmer productivity, quality of code, and migratability of code. It is expected that all but a small percentage of the code for ROMP processors will be written in PL.8.

Other programming aids include an assembly language simulator, a ROMP assembler, and linkage editor which assist the programmer in checking and documenting programs, in controlling address assignment, in segmenting a program, in data and symbol definition, in generating macro instructions, and in controlling the assembler itself.

"ROMP System Support Software" on page 184 contains a summary of support software status and lists available documentation.

1.5 HARDWARE DOCUMENTATION OVERVIEW

The ROMP Functional Specification is designed to provide an architectural description of the ROMP processor, and the ROMP Storage Channel. In addition, it is intended to provide a general description of how the ROMP processor can be employed in various system configurations, and to provide references to additional documentation concerning the detailed electrical and environmental characteristics of the ROMP chip, the ordering of internal registers in the ROMP processor, and a suggested support processor interface definition. The various documents describing these detailed aspects of the ROMP processor are included in "ROMP System Hardware References" on page 188.

1.6 SIGNAL NAMING CONVENTIONS

The terms active and inactive are used throughout this document to describe the state of various signals. All signal names (i.e. +DAL00, +EXCEPTION, -REQI0, etc.) are preceded by a + or -. If the signal name is preceded by a +, an active state is a voltage level of +2.4 volts or greater, and an inactive state is a voltage level of +0.8 volts or less. If the signal name is preceded by a -, an active state is a voltage level of +0.8 volts or less, and an inactive state is a voltage level of +2.4 volts or greater.

2.0 SYSTEM ORGANIZATION AND CONTROL

A ROMP system consists functionally of a ROMP , main storage, a bus converter, input/output devices, and possibly a direct memory access (DMA) controller, depending upon I/O device requirements. RSC devices may reference storage without processor involvement. This structure shown in Figure 1.

2.1 MAIN STORAGE

The ROMP system provides directly addressable main storage for data and instructions. The data units in main storage are shown in Figure 2 on page 6.

Up to 4.3 gigabytes of main storage may be directly addressed. Main storage is organized as a sequence of 32-bit words, each consisting of four 8-bit bytes. Bytes in main storage are consecutively numbered, left to right, starting with zero. Each number is considered the address of the corresponding byte. All addresses are computed as byte addresses. The address of a word has zeros in the two low-order bits. The address of a halfword has one zero in the low-order bit. Instructions must be located on halfword boundaries.

All storage effective addresses (base address plus displacement) are computed as 32-bit quantities. Wrap around is allowed and occurs on a 32-bit basis, i.e., main storage addressing wraps around from the architectural maximum byte address of 4,294,967,295 to address 0. This implementation of ROMP supports both 24-bit and 32-bit addressing. Systems which do not require virtual addressing can select the 24-bit addressing mode where the high-order byte of the 32-bit effective address is checked to be zero. A non-zero high-order byte in the effective address will cause a program check condition. Virtual address systems can select the 32-bit addressing mode which disables checking of the upper-byte.

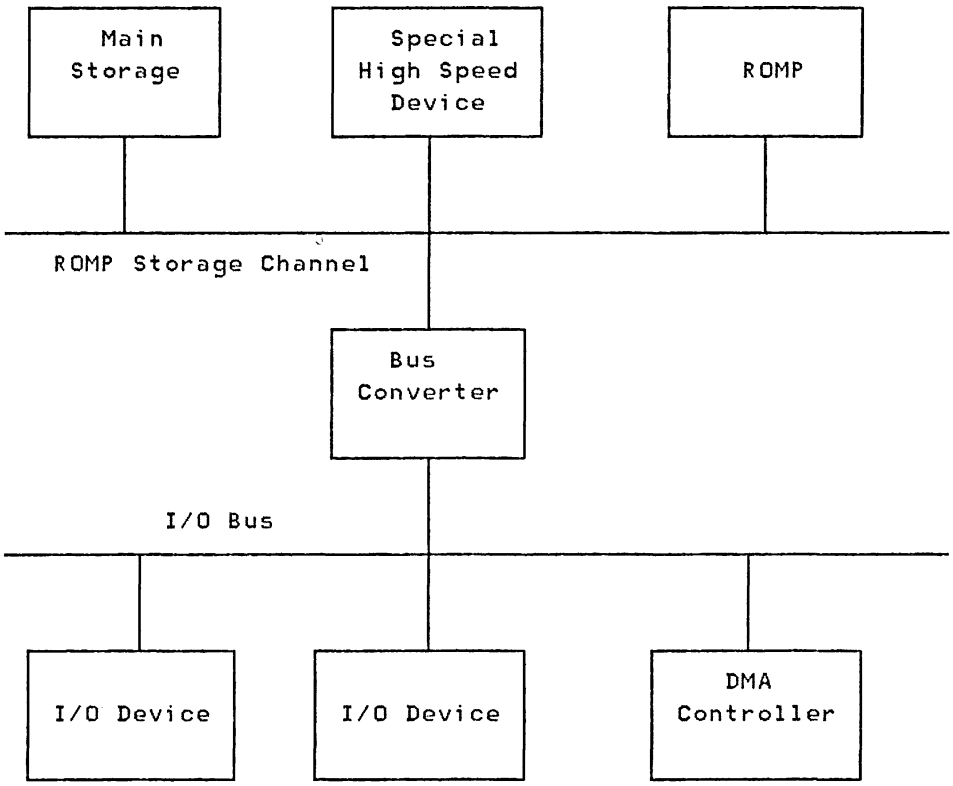


Figure 1. ROMP System

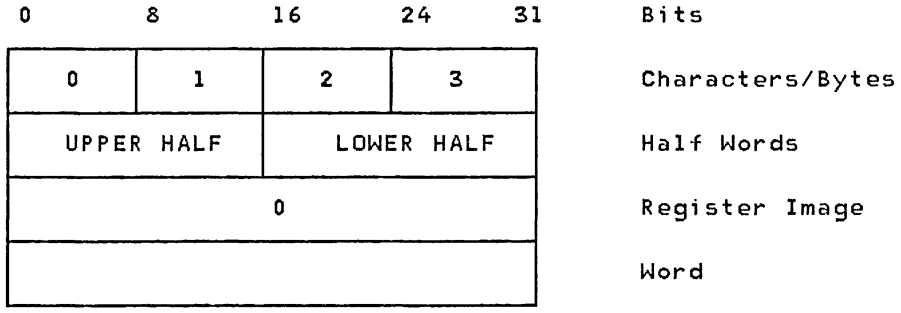


Figure 2. Data Units in Main Storage

2.2 STORAGE CHANNEL

ROMP's storage channel provides a 32-bit plus four parity bits wide synchronous bus which cycles at twice the processor cycle rate. The purpose of the channel is to enhance processor performance by having a bus dedicated to processor and storage transfers. All data and addresses transferred on the channel are multiplexed on the 32 address/data lines. An optional address extension bus consisting of 8 address lines plus parity is provided for systems using 32-bit addressing. Devices connected to the storage channel are identified by a five-bit tag identifier that devices put on the storage channel simultaneously with their request. Arbitration is accomplished by using a linear arbitration mechanism with devices connected in a daisy chain. For more information see "ROMP Storage Channel" on page 92.

2.3 PROGRAMMED I/O

Programmed I/O is allowed by the use of the Input/Output Read (IOR) and Input/Output Write (IOW) instructions. With these instructions, I/O devices can be read or written synchronously with program execution (see "Input/Output" on page 88).

2.4 PROCESSOR

The processor contains the sequencing and processing controls for instruction execution, interrupt action, the system timer and other control related functions.

Instructions are grouped into ten classes: storage access, address computation, branching, traps, moves and inserts, arithmetic, logical operations, shifts, system control, and input/output. A separate sub-section is devoted to each instruction class in "Instruction Set" on page 26.

2.5 PROCESSOR STATES

2.5.1 Executing Wait Check Stop and Stopped State

Four states of the processor are defined: executing state, wait state, check stop state, and stopped state.

The processor is in the executing state when it is executing instructions. In the executing state, instruction fetching and execution proceeds in the specified manner. Interrupts may occur between instructions as specified in "Interrupts" on page 17.

After the processor has executed the Wait instruction it is in the wait state. No other instructions are fetched or executed while the processor is in the wait state. The processor leaves the wait state and enters the executing state when an interrupt for which the processor is enabled occurs (see "Interrupts" on page 17). The instruction address in the old program status for the priority level associated with the interrupt contains the address of the instruction immediately following the Wait instruction. The I/O pin -WAIT is active when the processor is in the wait state.

When the processor is in the check stop state, no instructions are executed, interrupts do not occur, and system interface operations may be suspended. The processor enters the check stop state when one of the following occurs:

1. An error is detected during power-on diagnostics.

2. A machine check error is detected and the Check Stop Mask is zero.

3. A program check or machine check error is detected and the processor is servicing a machine check error.

4. A program check error is detected and the processor is servicing a program check error.

The check stop condition is cleared during a power-on reset. The processor machine check is described in "Machine-Check Errors" on page 121, and the program check is described in "Program-Check Errors" on page 124.

The stopped state is entered as a result of operator action from a control panel. Operator initiated load, display, and stepping functions occur in the stopped state as described in "Processor Support Functions" on page 142.

2.5.2 Problem and Supervisor States

The selection between problem and supervisor state determines whether the full set of instructions is valid. In supervisor state, all instructions are valid. In problem state, only instructions that cannot be used to affect system integrity are valid. The instructions that are not valid in problem state are called privileged instructions; they include those which inspect or modify any system control registers (except the Condition Status or Multiplier Quotient registers), the Load Program Status

instruction, and the Wait instruction. A privileged instruction encountered in the problem state constitutes a privileged instruction exception and causes a program check. "Privileged Instructions" on page 182 lists the privileged instructions.

The processor is in problem state when bit 21 of the Interrupt Control Status (ICS) is a one. The processor is in supervisor state when bit 21 of the ICS is a zero.

2.6 GENERAL-PURPOSE REGISTERS

The processor provides sixteen 32-bit general purpose registers (GPRs). All manipulation of data is performed in the GPRs.

Each GPR consists of an upper and lower half of sixteen bits each. The GPR may be partitioned into four eight-bit characters, C0, C1, C2, and C3. The general purpose register organization is shown in Figure 3 on page 10.

To avoid the destruction of operands in certain operations, some instructions cause the result of the operation to be placed in the twin of one of the GPR operands. The register twin of a given GPR has the name, in binary, of the given GPR with the low-order bit inverted. Thus the twin of GPR 5 (binary 0101) is GPR 4 (binary 0100), and the twin of GPR 14 (binary 1110) is GPR 15 (binary 1111). A register and its twin are referred to as a pair.

For computation purposes, the content of a GPR is treated as either a signed algebraic quantity, an unsigned positive quantity, or an unstructured logical quantity. In a GPR, an algebraic quantity is represented by 32 bits in two's complement form.

2.7 SYSTEM CONTROL REGISTERS

Sixteen 32-bit system control registers (SCR) exist in the processor. An entire SCR or fields within an SCR are assigned to particular facilities in the system such as interrupt, processor, and system timer. The register organization for SCRs is shown in Figure 4. Some SCRs and SCR fields are reserved and are not assigned to any system facility. The source bits are ignored on an attempt to set the reserved bits of an SCR. When the reserved bits of an SCR are fetched, the resulting values are unpredictable.

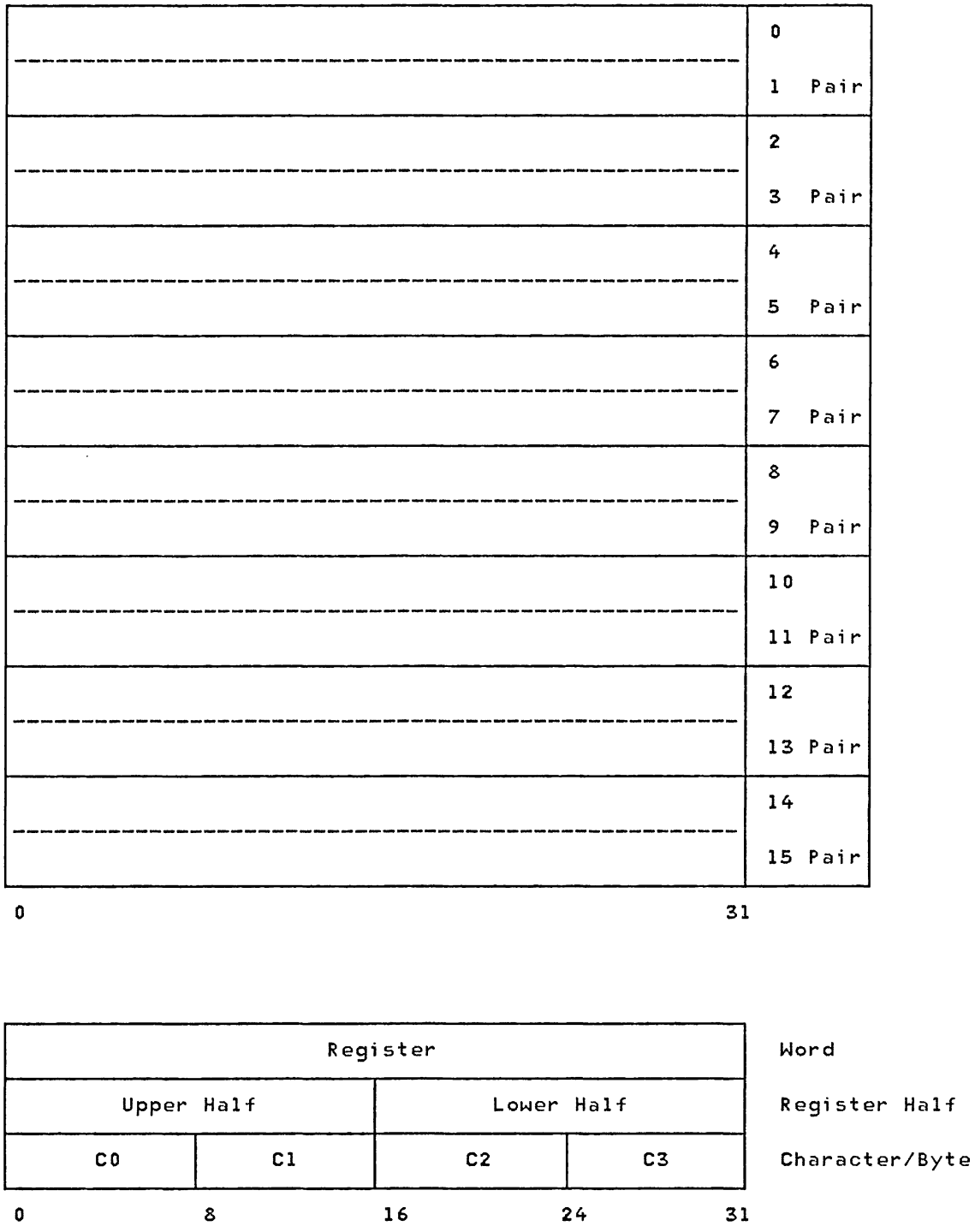


Figure 3. General Purpose Registers

Reserved				SCR 0
Reserved				SCR 1
Reserved				SCR 2
Reserved				SCR 3
Reserved				SCR 4
Reserved				SCR 5
Counter Source				SCR 6
Counter				SCR 7
Reserved			TS	SCR 8
Reserved				SCR 9
Multiplier Quotient				SCR 10
Reserved		MCS	PCS	SCR 11
Reserved		IRB		SCR 12
IAR				SCR 13
Reserved		ICS		SCR 14
Reserved		CS		SCR 15
0	8	16	24	31

COUS = Counter Source
COU = Counter
TS = Timer Status
MQ = Multiplier Quotient
MCS = Machine Check Status
PCS = Program Check Status
IRB = Interrupt Request Buffer
IAR = Instruction Address Register
ICS = Interrupt Control Status
CS = Condition Status

Figure 4. System Control Registers

2.7.1 Counter Source Counter and Timer Status

The Counter Source (COUS, SCR 6), Counter (COU, SCR 7), and Timer Status (TS, SCR 8) are necessary for the system timer operation and are described in "System Timer Facility" on page 14.

2.7.2 Multiplier Quotient

SCR 10 is referred to as the Multiplier Quotient (MQ). The MQ provides a GPR extension to accommodate the product for the Multiply Step instruction and the dividend for the Divide Step instruction (see "Divide And Multiply Step Instructions" on page 65).

2.7.3 Machine Check Status and Program Check Status

Bits 16-23 of SCR 11 are referred to as the Machine Check Status (MCS) and bits 24-31 are referred to as the Program Check Status (PCS). These two fields are used for error identification and are described in detail in "Machine-Check Errors" on page 121 and "Program-Check Errors" on page 124.

2.7.4 Interrupt Request Buffer

Bits 16-31 of SCR 12 are referred to as the Interrupt Request Buffer (IRB). The Interrupt Request Buffer allows interrupt requests to be generated under program control. For more detailed information refer to "Interrupt Request Buffer" on page 21.

2.7.5 Instruction Address Register

SCR 13 is referred to as the Instruction Address Register (IAR). The Instruction Address Register is a 32-bit register which contains the address of the next instruction to be executed. Since all instructions lie on halfword boundaries, the low-order bit (bit 31) of the Instruction Address Register is zero. Accesses for instructions may require the fetching of a word, a halfword, or the low-order halfword of a word followed by the high-order halfword of the next consecutive word in main storage.

Logically, during the execution of an instruction, the content of the Instruction Address Register is incremented by the length of the current instruction. Should this instruction be a successful branch/jump instruction, the content of the Instruction Address Register is changed to the address of the branch/jump target instruction. The IAR contains the address of the next instruction when the IAR is saved as part of the program status and when a system control instruction to read the IAR is executed.

2.7.6 Interrupt Control Status

Bits 16-31 of SCR 14 are referred to as the Interrupt Control Status (ICS). The ICS contains the Parity Error Retry Interrupt Enable bit, Storage Protect bit, Problem State bit, Translate Mode bit, Interrupt Mask, Check Stop Mask, Register Set Number, and the Processor Priority. The ICS is described in "Interrupt Control Status" on page 21.

2.7.7 Condition Status

Bits 16-31 of SCR 15 are referred to as the Condition Status (CS). The Condition Status contains information about the results of certain operations and provides a mechanism for decision making. The Condition Status is defined as follows:

Bits	16-23	Reserved
Bit	24	Permanent Zero (PZ)
Bit	25	Less Than (LT)
Bit	26	Equal (EQ)
Bit	27	Greater Than (GT)
Bit	28	Carry Zero (C0)
Bit	29	Reserved
Bit	30	Overflow (OV)
Bit	31	Test Bit (TB)

Bit 24 of the Condition Status is the Permanent Zero bit (PZ). It is set to zero whenever the Condition Status is loaded, and it cannot be set to one. Its presence provides for a guaranteed branch or jump by use of a branch on not condition bit or jump on not condition bit instruction specifying the Permanent Zero bit.

Bit 25 of the Condition Status is the Less Than bit (LT). This bit is set to one during logical, shift, and certain arithmetic instructions if the result is negative or if the high-order bit of the result is one; otherwise it is set to zero. It is also set during compare instructions to indicate the relative algebraic magnitudes of the comparands.

Bit 26 of the Condition Status is the Equal bit (EQ). This bit is set to one during logical, shift, and certain arithmetic instructions if all bits of the result are zeros; otherwise it is set to zero. It is also set during compare instructions if the comparands are equal.

Bit 27 of the Condition Status is the Greater Than bit (GT). This bit is set to one during logical, shift, and certain arithmetic instructions if the sign bit of the result is zero and the result is nonzero; otherwise it is set to zero. It is also set during compare instructions to indicate the true relative algebraic magnitudes of the comparands.

Bit 28 of the Condition Status is the Carry Zero bit (C0). This bit is set to one during certain arithmetic instructions if the operation generates a carry out of bit position zero; otherwise it is set to zero.

Bit 29 of the Condition Status is a reserved bit.

Bit 30 of the Condition Status is the Overflow bit (OV). It is set to one during certain arithmetic instructions if the signed result of the operation cannot be represented in 32 bits; otherwise it is set to zero.

Bit 31 of the Condition Status is the Test bit (TB). It is set by the move to test bit instructions, where a specified bit of the half of a register is moved to the test bit. It is also affected by instructions which load or directly alter the Condition Status register.

All bits of the Condition Status, except the Permanent Zero bit, can be set through use of the move to SCR instructions.

A four-bit field in the conditional branch instructions specifies the Condition Status bit to be tested. A zero in the four-bit field of a branch instruction specifies bit 16 of the Condition Status, a one specifies bit 17 of the Condition Status, and so on. A three-bit field in the conditional jump instructions specifies the Condition Status bit to be tested. A zero in this three-bit field specifies bit 24, a one specifies bit 25, and so on.

2.8 SYSTEM TIMER FACILITY

Many applications require a knowledge of real time for such functions as system counting, time slicing, time stamping, interval timing, and timing the productivity of operations. These functions can be provided using the system timer facility.

For some devices, the device requirements may be such that additional timers are needed in the adapter. A more sophisticated timer can be provided as an I/O device if needed.

System timer updating occurs at the frequency of the clock connected to the timer clock I/O pin.

This section describes the system timer facility and its operation.

2.8.1 Counter

SCR 7 is referred to as the Counter (COU) and is a thirty-two-bit count-down counter. The Counter is decremented from an external source connected to the I/O pin -TIMER CLOCK. The counter is updated on an inactive to active transition of -TIMER CLOCK. Processor instruction execution is suspended during the counter update. When the Counter is decremented from 1 to 0, the value contained in the Counter Source is loaded into the Counter and the alarm action is initiated. This action is such that normal operations will continue by the time the next count pulse arrives. The alarm action is to set a bit in the IRB whose priority level corresponds to the Timer Interrupt Priority in SCR 8, if the timer is enabled. The alarm also sets the Interrupt Status bit to one, and updates the Timer Status. The contents of the Counter Source are not altered.

2.8.2 Counter Source

The Counter Source (COUS), SCR 6, consists of the thirty-two-bit value that is automatically loaded into the Counter when an alarm occurs.

2.8.3 Timer Status

The Timer Status (TS), bits 24-31 of SCR 8, is defined as follows:

Bit 24 Reserved.

Bit 25 Enable. When zero, no interrupts are created. This does not start or stop the counter, but enables/disables the setting of IRB bits. At power-on reset, this bit is zero.

Bit 26 Interrupt Status. When one, an alarm has occurred. This bit is set only if an alarm has occurred, and the Enable bit is set to one. This bit is reset by software when the counter is serviced. Software can reset this bit by executing a Clear SCR Bit (CLRSB) instruction.

Bit 27 Overflow. When one, more than one alarm has occurred before the Interrupt Status bit has been reset. This bit is also reset by software when the counter is serviced.

Bit 28 Reserved

Bits 29-31 Timer Interrupt Priority. A timer alarm causes the setting of an IRB bit corresponding to the priority level specified by this field if the timer is enabled.

2.8.4 Programming Note: System Timer Operation

To provide an interval timer, the Counter is directly loaded with a value corresponding to the amount of time until the interval is to expire.

To provide a fixed interval interrupt, an appropriate value is loaded into the Counter Source and then not changed. For example, if the Counter Source was loaded with 5 (X'05'), and a 1 millisecond timer clock was used, the processor would be interrupted every 5 milliseconds; if it was loaded with 250 (X'FA'), the processor would be interrupted every one-fourth second. Software could then update internal storage locations and provide time-of-day in whatever format desired.

Note that loading the Counter Source does not alter the value in the Counter. As a result, the interrupt interval corresponding to the value loaded into the Counter Source will not begin until the Counter is decremented from 1 to 0, and the new Counter Source is loaded into the counter. To synchronize the Counter with a new Counter Source, both the Counter Source and the Counter must be loaded with the new Counter Source value.

Multiple simultaneous timings can be handled using the system timer as a resource. The Counter is loaded from a queue whose entries are calculated to be the "time" from the completion of the previous entry until the time for the entry in question to be completed.

The external clock allows the system timer to be used to count external events and can notify the program when a specific count

arrives. This source need not provide a regular, clock-like signal; it can be either a regular or irregular source.

The value loaded into the Timer Interrupt Priority (TS, bits 29-31) must be greater than or equal to zero (000) and less than or equal to six (110). These are the only values for which a corresponding IRB exists. A value of seven (111) in the Timer Interrupt Priority will cause no bit in the IRB to be set when a timer interrupt occurs.

2.9 INTERRUPTS

The Interrupt facility permits the processor to change its status at the request of some other system component or due to processor conditions established by the program. Interrupt processing consists of saving the current program status and establishing the program status for servicing the interrupt. Interrupts only occur on instruction boundaries, but some instruction sequences are not interruptible. A Load Program Status (LPS) instruction is provided for software to return from an interrupt. Execution of a LPS instruction restores the IAR, the CS, and the ICS to the values that existed when the interrupt occurred (see "Load Program Status Instruction" on page 85).

The processor may also change its status as a result of error conditions within the processor or a system component. Error processing consists of saving the current program status and establishing the program status for servicing the error. Errors are grouped into two classes: Machine check errors and program check errors. These errors are discussed in detail in "Reliability, Availability, and Serviceability" on page 121.

The interrupt facility is a priority-based mechanism. This permits the servicing of higher priority functions to take precedence over the servicing of lower functions.

Interrupt sources consist of the seven external interrupt inputs (-REQI0-6), software interrupts posted via setting of bits in the IRB, and error conditions (either the -TRAP input, or internal errors) detected during system operation. The seven external interrupt inputs and software setting of IRB interrupt request bits are treated in the same manner by ROMP The interrupt request level is compared to the current processor priority specified by bits in the ICS. If the interrupt request represents a higher priority than the current processor priority, and interrupts are enabled by the Interrupt Mask in the ICS, then the interrupt is taken.

Taking an interrupt consists of saving the current processor status in the old PSW corresponding to the level of the interrupt request, and loading a new processor status from the new PSW

corresponding to the level of the interrupt request. Saving of the current processor status requires saving the address of the instruction, the condition status, and the ICS when the interrupt occurred. Loading of the new processor status requires loading the new IAR (containing the address of the interrupt service routine) and the new ICS from the new PSW. Saving of the current processor status and loading of the new processor status is performed automatically by ROMP hardware. Note that none of the GPRs are automatically saved by hardware. Software is responsible for saving any GPRs modified by the interrupt service routine. Once the interrupt has been serviced, execution of the old program can be resumed by loading the old program status word via an LPS instruction. This will restore the IAR, the CS, and the ICS to the values that existed when the interrupt occurred.

In addition to the seven interrupt levels, the detection of error conditions can cause interrupts to the Program Check and Machine Check interrupt levels. Interrupts to the Program Check level consists of errors which are most probably due to software errors (i.e. detection of an invalid op-code, addressing error, detection of a privileged instruction exception, etc.). Interrupts to the Machine Check level consists of errors which are most probably due to hardware errors (i.e. RSC parity errors, and RSC timeouts). In addition, an external input (-TRAP) can be used by system components to cause a machine check interrupt. "Machine-Check Errors" on page 121 and "Program-Check Errors" on page 124. describe Machine Check and Program Check interrupts.

2.9.1 Processor Priority

Under normal system conditions, the processor executes instructions at a level of priority referred to as the Processor Priority. The Processor Priority may assume one of eight levels as specified by a three-bit field in the ICS. Priorities for the eight levels are represented by the following inequality:

Priority of Level 0 > Priority of Level 1 >...> Priority of Level 7

The Processor Priority may be changed either by an interrupt or by an instruction which modifies the Processor Priority. There are two sources of interrupts: an interrupt condition signaled via the Interrupt Request Buffer, or an interrupt condition signaled by some system component via the seven interrupt request inputs (-REQI0-6).

The processor may also execute instructions at two levels which are not accessible via the interrupt facility. These levels are provided for the reporting and servicing of machine check and program check error conditions as discussed in "Machine-Check

Error Handling" on page 122 and "Program-Check Error Handling" on page 124.

2.9.1.1 Interrupt Request Priority

Interrupt requests occur on one of seven priority levels. Priorities for the seven levels are represented by the following inequality:

Priority of level 0 > Priority of Level 1 >...> Priority of Level 6

The processor may execute instructions with a processor priority of 7, but no interrupt requests with a priority of seven can occur.

2.9.1.2 Interrupt Priority Assignment

A bit being set to one in the Interrupt Request Buffer causes an interrupt request to the level corresponding to that bit. Timer interrupts cause an interrupt request (via the IRB) to the level specified in the Timer Status. A system component causes an interrupt request at a level determined by the attachment of its Interrupt Request line (-REQI0 through 6) to the processor.

2.9.2 Point of Interrupt

Interrupts only occur on instruction boundaries. Furthermore, interrupts are prevented from occurring within certain instruction sequences. A branch with execute instruction and its subject instruction are uninterruptable.

Thus, a branch with execute and its subject instruction is considered to be a unit, and interrupts only occur before or after the unit is executed (refer to "Instruction Set" on page 26).

2.9.3 Error Handling

If the processor is executing an error routine as a result of a machine check or program check error condition, all interrupt requests from system components and interrupts signaled via the IRB remain pending.

2.9.4 Program Status

The state of the processor is called the program status. The program status consists of the contents of the following system control registers: the Instruction Address Register, the Condition Status, and the Interrupt Control Status.

Upon interrupt the current program status is automatically saved in the old program status location. The program status for servicing the interrupt is loaded from the new program status location, with the exception of the Condition Status. The Condition Status is not changed by loading the new program status.

2.9.4.1 Old/New Program Status Pairs

An old/new program status pair consists of eight bytes of old program status and eight bytes of new program status. When an interrupt occurs, the old/new program status pair is specified by the priority level of the interrupt.

2.9.4.2 Location of Old/New Program Status Pairs

The program status save area in main storage contains ten old/new program status pairs. Two old/new program status pairs are for machine check and program check error handling, one old/new program status pair is used for the Supervisor Call (SVC) instruction, and the remaining seven are for interrupt servicing. The structure of an old/new program status pair is shown in Figure 5 on page 24. Note that the SVC old/new program status pair contains a sixteen-bit SVC interrupt code which is generated by execution of the SVC instruction (See "Supervisor Call Instruction" on page 87). Figure 6 on page 25 shows the organization of the program status save area in main storage. The 16-byte old/new program status pairs are located in consecutive main storage locations. The program status save area is located at addresses X'100' thru X'19F'. Address translation is disabled for storing of the old program status and loading of the new program status.

2.9.5 System Control Registers

Two fields are provided within the system control registers to support the interrupt facility. They are the Interrupt Request Buffer and the Interrupt Control Status.

2.9.5.1 Interrupt Request Buffer

The Interrupt Request Buffer (IRB) is a 16-bit field in SCR 12 and has the following format:

Bit 16	Interrupt Request Level 0
Bit 17	Interrupt Request Level 1
Bit 18	Interrupt Request Level 2
Bit 19	Interrupt Request Level 3
Bit 20	Interrupt Request Level 4
Bit 21	Interrupt Request Level 5
Bit 22	Interrupt Request Level 6
Bits 23-31	Reserved

The IRB provides the capability of generating interrupt requests under software control. Setting an IRB bit to one causes an interrupt request to the level corresponding to that bit. The interrupt request remains active until the bit is cleared by software.

If bit 25 of SCR 8 is one (enabled), a timer alarm caused by the Counter being decremented from 1 to 0 sets a bit in the IRB, which generates an interrupt request. The bit in the IRB which is set is determined by the Timer Interrupt Priority in the Timer Status.

2.9.5.2 Interrupt Control Status

The Interrupt Control Status (ICS) is a 16-bit field in SCR 14 with the following format:

Bits 16-18	Reserved
Bit 19	Parity Error Retry Interrupt Enable
Bit 20	Storage Protect
Bit 21	Problem State
Bit 22	Translate Mode
Bit 23	Interrupt Mask
Bit 24	Check Stop Mask
Bits 25-27	Register Set Number
Bit 28	Reserved
Bits 29-31	Processor Priority

A value of one in the Parity Error Retry Interrupt Enable bit enables interrupts when a ROMP Storage Channel (RSC) retry successfully completes a processor generated transfer that was previously unsuccessful due to detection of a parity error. A successful parity error retry interrupt will cause a level 0 interrupt by setting the Interrupt Request Level 0 bit (bit 16) in the Interrupt Request Buffer. The RSC Check bit (bit 16) in the Machine Check Status will be set to indicate the cause of the interrupt.

A value of one in the Storage Protect bit enables address checking in the storage controller. Use of Storage Protect is described in "Storage Protect" on page 137.

A value of one in the Problem State bit places the processor in problem state; a value of zero places the processor in supervisor state.

A value of one in the Translate Mode bit enables address translation in the storage controller. Use of Translate Mode is described in "Address Translation" on page 140.

A value of one in the Interrupt Mask inhibits all system component, timer, and software interrupts on all levels. An interrupt which is inhibited remains pending.

A value of one in the Check Stop Mask prevents the processor from entering the Check Stop state upon the detection of a machine check error.

The three-bit encoded Register Set Number allows one of eight register sets to be specified as the active register set. The current ROMP design implements one of the eight register sets. Bits 25-27 are ignored in this implementation.

The three-bit encoded Processor Priority indicates the current processor priority level. Interrupt requests with priorities lower than or equal to the current processor priority are ignored.

2.9.6 Occurrence of Interrupts

An interrupt occurs due to a bit in the IRB being equal to one if the Processor Priority is lower than the priority corresponding to that bit of the IRB, the Interrupt Mask is zero, and no system component is signaling an interrupt request on a higher level than that signaled via the IRB.

An interrupt occurs due to a system component interrupt request if the Processor Priority is lower than that of the interrupt request, the Interrupt Mask is zero, and the IRB is not signaling an interrupt request on a higher level than that signaled by the system component.

2.9.7 Programming Note: Interrupt Facility

The interrupt facility contains features which, if used improperly, may force the processor into an infinite hardware

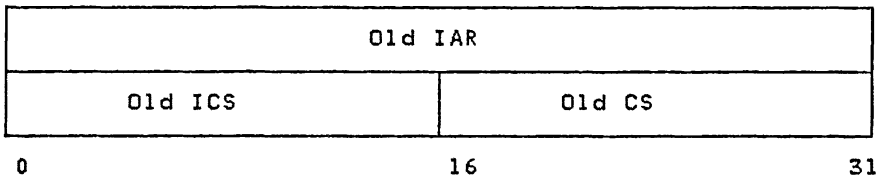
loop. When the processor loads the new program status for servicing an interrupt, it loads the Processor Priority from the ICS in the new program status location. The value in the Processor Priority in the new program status is completely under software control. This processor priority which is loaded must not be lower than the priority of the interrupt request which caused the interrupt, if the Interrupt Mask in the new program status is zero, the same interrupt request will immediately cause another interrupt. Multiple interrupts would continue to occur until a system component signals an interrupt request on a higher level, or a power-on reset occurs.

2.9.8 Programming Notes: Interrupt Servicing

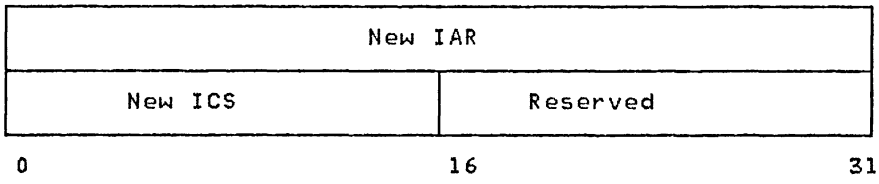
The program should issue an IOW instruction to signal the device that the interrupt request is being serviced, and to reset the interrupt request bit in the device status.

1. The program should clear the interrupt request of the interrupting device as soon as possible after the point of interrupt. This allows the processor to determine the priority of the next highest interrupt request.

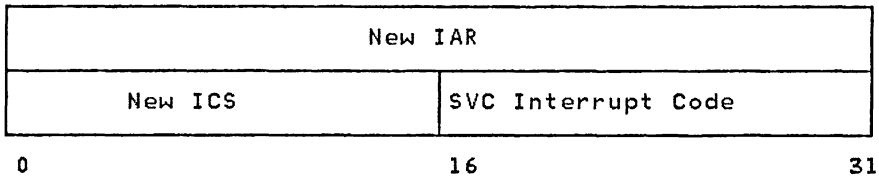
2. A Load Program Status (LPS) instruction is provided for software to return from an interrupt (see "Load Program Status Instruction" on page 85). The effective address of the LPS instruction points to the Program status to which control is being returned. Normally control will be returned to the previously active program whose program status is located in the old program status associated with the interrupt being serviced.



Old Program Status



New Program Status

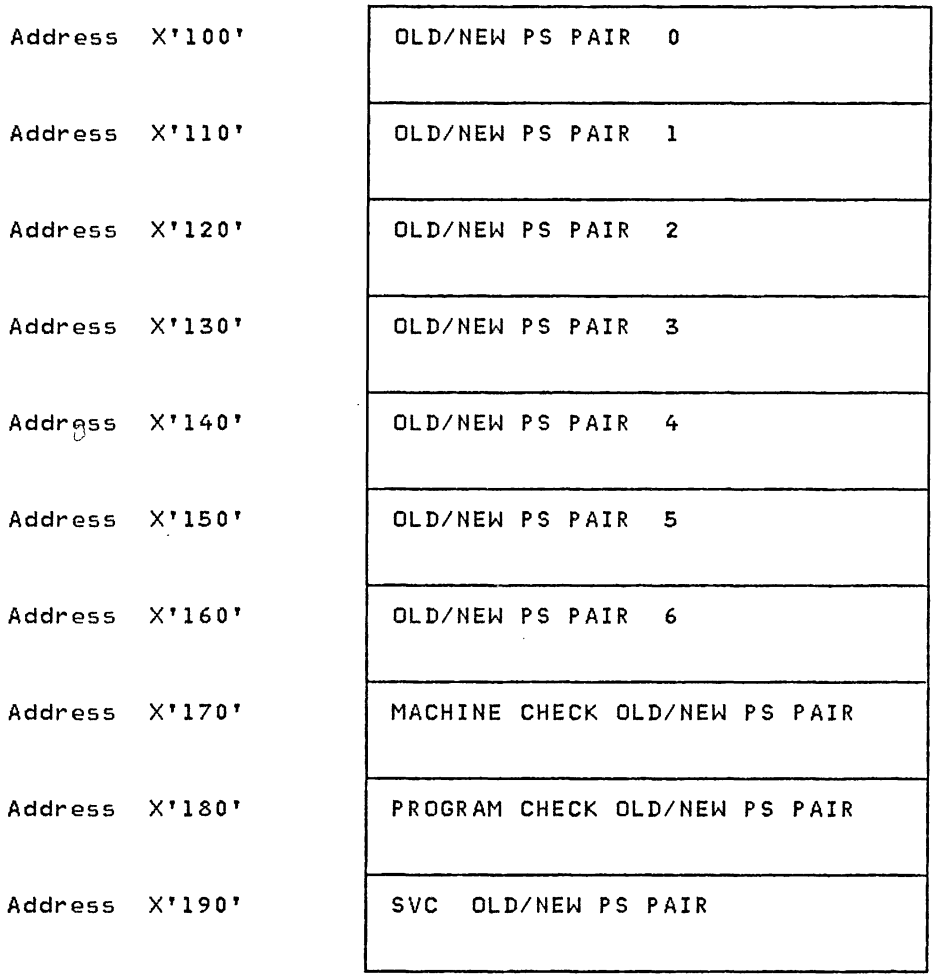


SVC New Program Status

IAR = Instruction Address Register
ICS = Interrupt Control Status
CS = Condition Status
SVC = Supervisor Call

Note: Reserved bits in the old program status are set to unpredictable values. Reserved bits in the new program status are ignored.

Figure 5. Old/New Program Status Pair



Main Storage Addresses X'100' Through X'19F'

Note: Each PS pair requires 16 bytes.

Figure 6. Program Status Save Area

3.0 INSTRUCTION SET

3.1 GENERAL DESCRIPTION

Instructions are grouped into ten classes: storage access, address computation, branching, traps, moves and inserts, arithmetic, logical operations, shifts, system control, and input/output. A separate section is devoted to each instruction class. Each instruction is specified in terms of mnemonic, operation code (op code), length, and functional description.

Unassigned op codes are reserved for future use. If these reserved op codes are encountered by the processor, a program check error occurs. For more detailed information, see "Program-Check Errors" on page 124.

The ROMP processor does not support dynamic instruction modification. Any attempt by software to modify an instruction may result in unpredictable operation.

ROMP provides a supervisor state in which all instructions are valid, and problem state in which only instructions that cannot be used to affect system integrity are valid. The instructions that are not valid in problem state are called privileged instructions. A privileged instruction encountered in the problem state constitutes a privileged instruction exception and causes a program check.

The following notation is used to describe each instruction:

GPR	General Purpose Register (The word register is also used to to denote a GPR)
SCR	System Control Register
IAR	Instruction Address Register
IRB	Interrupt Request Buffer
MCS	Machine Check Status
PCS	Program Check Status
CS	Condition Status
ICS	Interrupt Control Status

RA,RB or RC	These abbreviations denote fields in the instruction which specify GPRs.
SRB	This denotes a field in the instruction which specifies an SCR.
I	This denotes a field of immediate data in the instruction.
N	This denotes a Condition Status bit number.
JI	This denotes an eight-bit relative branch displacement in the JI format instructions.
BI	This denotes a 20-bit relative branch displacement in the BI format instructions.
BA	This denotes a 24-bit absolute branch address.
0/(RC)	This indicates the value 0 if RC is specified as 0, else the content of register RC. (i.e. if the RC field is specified as 0, a value of zero is used for the computation, if the RC field is not 0, the content of the specified register is used for the computation.) Register 0 can not be used as the RC register.
0[n]	This indicates a field of zeroes, n bits wide.
//	Two parallel bars are used to indicate a concatenation of the two fields specified on either side of the bars.
(RC)	A register specification enclosed in parentheses indicates the content of the specified register.

The seven instruction formats (JI, X, D-Short, R, BI, BA and D) are shown in Figure 7. Instructions are either two or four bytes in length. The first four, five or eight bits of an instruction are referred to as the operation code (op code). The JI format has a five bit op code. The X and D-Short formats both have four-bit op codes. The R, BI, BA and D formats all have eight-bit op codes. Instructions of formats JI, X, D-Short and R are all two bytes long. Instructions of formats BI, BA and D are all four bytes long.

The RA, RB and RC fields specify GPRs. The SRB field specifies an SCR. The I field specifies a displacement of a storage address or an immediate value. The N field specifies a Condition Status bit. Relative branch displacements JI and BI are both signed binary numbers in two's complement form, while BA designates an absolute branch address

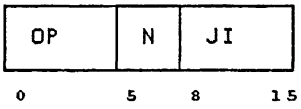
Some R format instructions have an SRB, I, or N field instead of an RB or RC field.

For X, D-Short, and D format instructions which refer to main storage or system components, the address is calculated according to the following formulas:

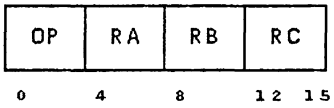
X Format	(RB) + 0/(RC)
D-Short Format	0/(RC) + 0[28]//I
	0/(RC) + 0[27]//I//0[1]
	0/(RC) + 0[26]//I//0[2]
D Format	0/(RC) + 0[16]//I
	0/(RC) + Sign Extended I

Where 0/(RC) indicates the value 0 if RC is specified as 0, and the value of the content of the general purpose register if RC is specified as nonzero.

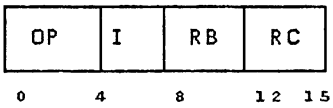
JI Format



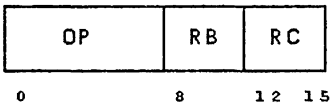
X Format



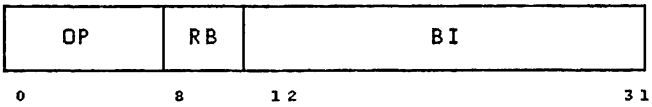
D-Short Format



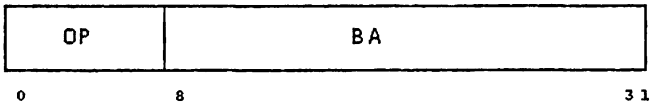
R Format



BI Format



BA Format



D Format

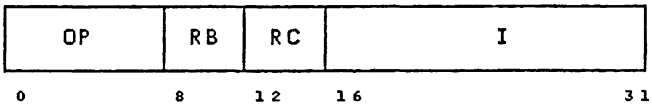


Figure 7. Instruction Formats

3.2 STORAGE ACCESS

Main storage is organized as a sequence of eight-bit bytes with a maximum capacity of 4,294,967,296 bytes. All storage effective addresses (base address plus displacement) are computed as 32-bit quantities. Wrap around is allowed and occurs on a 32-bit basis, i.e., main storage addressing wraps around from the architectural maximum byte address of 4,294,967,295 to address 0. This implementation of ROMP supports both 24-bit and 32-bit addressing. In 24-bit addressing mode, the high order 8 bits of the 32-bit effective address is checked to be zero. A non-zero high order byte in the effective address will result in a program check. If less than the maximum amount of main storage is installed, an attempt to utilize a byte from a non-existent main storage location will result in a program check.

All storage accesses are for a byte or multiples thereof. Instructions are provided to load or store a single character, a halfword, or a word into a general-purpose register. Storage accesses for halfwords and words ignore the low-order bit or pair of bits, respectively, of the effective address. The address of a halfword or word in main storage is the address of its leftmost byte. The Condition Status is not changed by any of these instructions.

A storage access to an invalid storage location will set the data address exception bit in the program check status and result in a program check. Refer to "Program-Check Errors" on page 124 for a description of the program check status.

All storage access instructions are non-privileged.

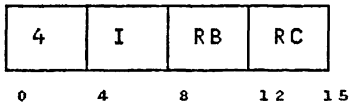
Engineering Note: Data Alignment

Data alignment for halfword and fullword accesses is normally provided in the storage controller by ignoring the low-order address bit for halfword accesses and the two low-order bits for fullword accesses. The effective storage address computed by ROMP for halfword and fullword data accesses is not aligned (i.e. the storage address is the byte address of the leftmost byte of the halfword or fullword). This allows a storage controller to support unaligned halfword and fullword accesses, if required in a particular system.

3.2.1 Load Instructions

Load Character Short D-Short Format

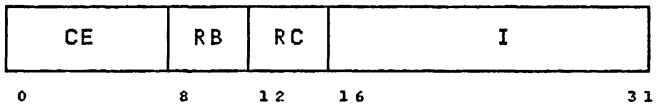
LCS RB, I(RC)



Character C3 of register RB is replaced by the character of storage addressed by 0/(RC) + 0[28]//I. Character C0 through C2 of register RB are set to zeroes.

Load Character D Format

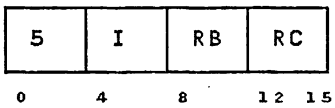
LC RB, I(RC)



Character C3 of register RB is replaced by the character of storage addressed by 0/(RC) plus the sign extended I-field. Character C0 through C2 of register RB are set to zeroes.

Load Half Algebraic Short D-Short Format

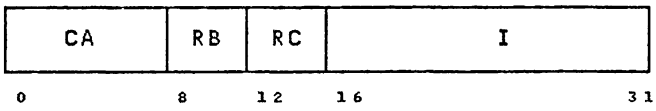
LHAS RB, I(RC)



The lower half of register RB is replaced by the halfword of storage addressed by 0/(RC)+0[27]//I//0[1]. The sign bit of the addressed halfword is extended through the upper half of register RB.

Load Half Algebraic D Format

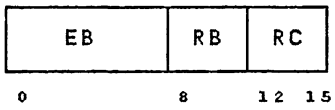
LHA RB, I(RC)



The lower half of register RB is replaced by the halfword of storage addressed by 0/(RC) plus the sign extended I-field. The sign bit of the addressed halfword is extended through the upper half of register RB.

Load Half Short R Format

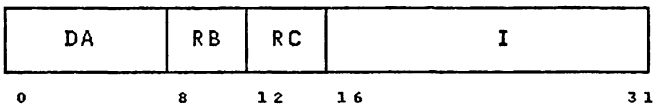
LHS RB, 0(RC)



The lower half of register RB is replaced by the halfword of storage addressed by the content of register RC. The upper half of register RB is set to zero.

Load Half D Format

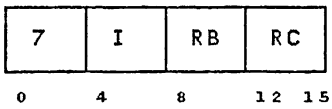
LH RB, I(RC)



The lower half of register RB is replaced by the halfword of storage addressed by 0/(RC) plus the sign extended I-field. The upper half of register RB is set to zeroes.

Load Short D-Short Format

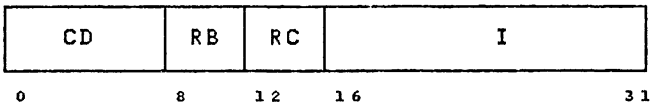
LS RB, I(RC)



The content of register RB is replaced by the word in storage addressed by 0/(RC) + 0[26]//I//0[2].

Load D Format

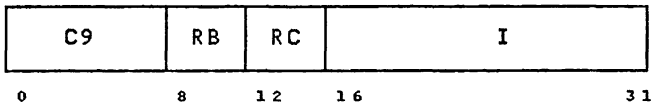
L RB, I(RC)



The content of register RB is replaced by the word in storage addressed by 0/(RC) plus the sign extended I-field.

Load Multiple D Format

LM RB, I(RC)

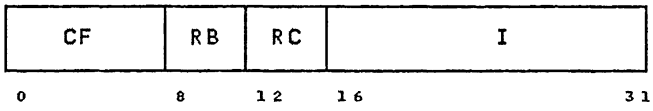


The content of registers RB through 15 are replaced, respectively, by the consecutive words in storage beginning at the address given by 0/(RC) plus the sign extended I-field.

3.2.2 Test and Set Instruction

Test and Set Half D Format

TSH RB, I(RC)



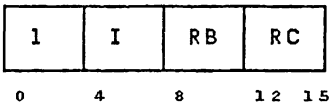
0

The lower half of register RB is replaced by the halfword of storage addressed by 0/(RC) plus the sign extended I-field. The upper half of register RB is set to zeroes. Immediately following the read operation, the storage unit will write all 1's in the high order byte of the selected halfword without permitting any other storage operations between the read and the write. The low-order byte of the selected halfword is left unaltered.

3.2.3 Store Instructions

Store Character Short D-Short Format

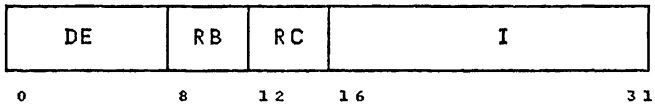
STCS RB, I(RC)



The character of storage addressed by 0/(RC) + 0[28]//I is replaced by character C3 of register RB.

Store Character D Format

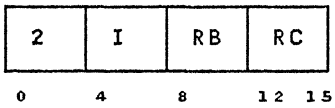
STC RB, I(RC)



The character of storage addressed by 0/(RC) plus the sign extended I-field is replaced by character C3 of register RB.

Store Half Short D-Short Format

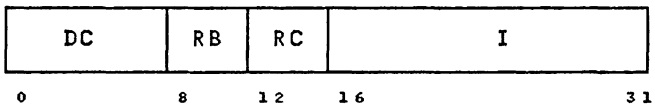
STHS RB, I(RC)



The halfword of storage addressed by 0/(RC) + 0[27]//I//0[1] is replaced by the lower half of register RB.

Store Half D Format

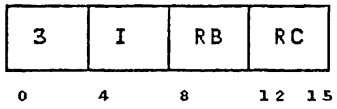
STH RB, I(RC)



The halfword of storage addressed by 0/(RC) plus the sign extended I-field is replaced by the lower half of register RB.

Store Short D-Short Format

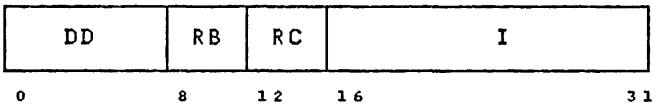
STS RB, I(RC)



The word of storage addressed by 0/(RC) + 0[26]//I//0[2] is replaced by the content of register RB.

Store D Format

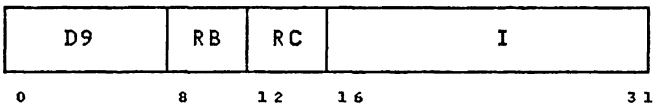
ST RB, I(RC)



The word in storage addressed by 0/(RC) plus the sign extended I-field is replaced by the content of register RB.

Store Multiple D Format

STM RB, I(RC)



The consecutive words in storage beginning at the address given by 0/(RC) plus the sign extended I-field are replaced, respectively, by the content of registers RB through 15.

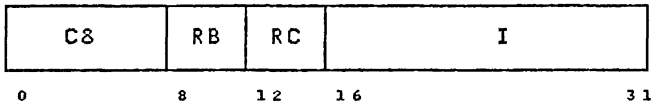
3.3 ADDRESS COMPUTATION

The address computation instructions operate only on the contents of the general purpose registers. No storage references for operands occur. The resultant values are not inspected for address exceptions. The Condition Status is not changed by any of these instructions.

All address computation instructions are non-privileged.

Compute Address Lower Half D Format

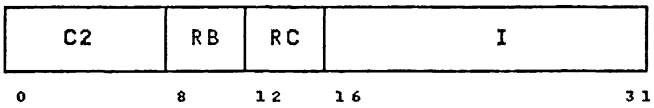
CAL RB, I(RC)



The address specified by 0/(RC) plus the sign extended I-field replaces the content of register RB.

Compute Address Lower Half 16-Bit D Format

CAL16 RB, I(RC)



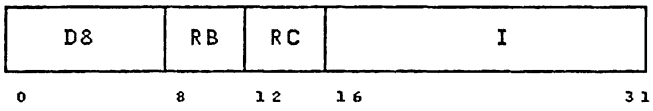
The 16-bit address specified by 0/(RC)[16:31] + I replaces the content of register RB[16:31], and 0/(RC)[0:15] replaces the content of register RB[0:15].

Programming Note:

This instruction is provided to assist in simulation of 16-bit architectures.

Compute Address Upper Half D Format

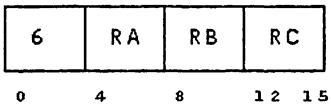
CAU RB, I(RC)



The address specified by 0/(RC) + I//0[16] replaces the content of register RB.

Compute Address Short X Format

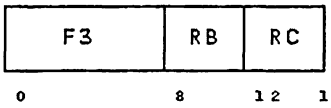
CAS RA,RB,RC



The address specified by (RB) + 0/(RC) replaces the content of register RA.

Compute Address 16-Bit R Format

CA16 RB, RC



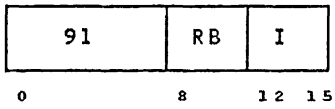
The 16-bit address specified by (RB)[16:31] + (RC)[16:31] replaces the content of register RB[16:31], and (RC)[0:15] replaces the content of register RB[0:15].

Programming Note:

This instruction is provided to assist in simulation of 16-bit architectures.

Increment R Format

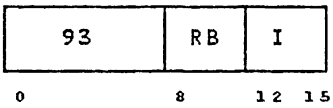
INC RB, I



The field I, extended on the left with 28 zeroes, is added to the content of register RB and the result placed into register RB.

Decrement R Format

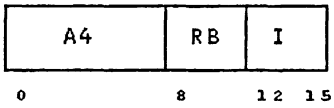
DEC RB, I



The field I, extended on the left with 28 zeroes, is subtracted from the content of register RB and the result placed into the register RB.

Load Immediate Short R Format

LIS RB, I



The content of register RB is replaced by field I, extended on the left with 28 zeroes.

3.4 BRANCHING

The normal sequential execution of instructions may be changed by the use of the branch instructions. These instructions permit subroutine linkage, decision making, and loop control, and provide several different target addressing forms.

For every branch instruction, except jumps, there is a corresponding branch with execute instructions. The instruction immediately following a branch with execute is called the subject instruction, and it is executed regardless of the branch decision, as if it preceded the branch. However, the subject instruction cannot affect the branch decision. Any Condition Status changes caused by the subject instruction occur after the branch decision has been made.

Subroutine linkage is provided by the branch and linkage instructions: BALA, BALAX, BALI, BALR, and BALRX. These instructions cause a branch to a new instruction sequence but preserve a return address in an implicitly or explicitly designated general purpose register. For the nonexecute forms of the instructions, the return address is the updated instruction address, which is the address of the halfword immediately following the branch and link instruction in storage. For the execute forms of the instructions, the return address is the address of the halfword which is four bytes beyond the end of the branch and link with execute instruction, i.e., it is the updated instruction address plus four. This allows four bytes following the branch and link with execute for the subject instruction. If the subject instruction requires only two bytes, the two remaining bytes are ignored.

Decision making and loop control are provided by the conditional branch and conditional jump instructions: BB, BBX, BBR, BBRX, BNB, BNBX, BNBR, BNBRX, JB, and JNB. For the conditional branch instructions, the branch decision is based on any specified state of any bit of the Condition Status. In these instructions, the value of N specifies the Condition Status bit with CS bit 16 specified by a value of 0, CS bit 17 by a value of 1, and so forth. For the conditional jump instructions the branch is based on any specified state of the rightmost eight bits (bits 24-31) of the Condition Status. In this case, the value of the N field of the jump instruction specifies the Condition Status bit with CS bit 24 specified by a value of 0, CS bit 25 by a value of 1, and so forth. For conditional branch instructions, the branch decision is based on any specified state of the rightmost sixteen bits (bits 16-31) of the Condition Status. In this case, the value of the N field specifies which Condition Status bit is used for the branch decision, with CS bit 16 specified by a value of 0, CS bit 17 specified by a value of 1, and so forth. If a reserved bit in the Condition Status (bits 16-23) is specified, the branch decision is unpredictable.

The branch instructions provide three different branch target addressing forms: absolute, absolute immediate, and relative immediate. The instructions BALR, BALRX, BBR, BBRX, BNBR, and BNBRX are absolute instructions and specify as an operand a register which contains the 24-bit branch target address. The instructions BALA and BALAX are absolute immediate instructions, where the full 24-bit branch target address is contained in the instruction. The instruction BALI, BALIX, JB, BB, BBX, JNB, BNB, and BNBX are relative immediate instructions; each contains an immediate field which is sign extended, logically shifted left one bit, and added to the address of the branch instruction in order to calculate the branch target address. The jump instructions (JB and JNB) contain an eight-bit immediate field which allows a jump range of -127 to +128 halfwords from the jump instruction. The branch instructions BALI, BALIX, BB and BBX contain a 20-bit immediate field which allows a branch range of -524,286 to +524,289 halfwords from the branch instruction.

The branch with execute instruction and its subject instruction are considered to be a single instruction. Thus, interrupts are not honored between the execution of the branch with execute instruction and the execution of its subject instruction.

Certain instructions are not allowed to be the subject of a branch with execute instruction. Since the branch with execute instructions change the normal sequential execution of instructions, the subject instruction cannot be an instruction which also changes the instruction sequencing, or the processor may be put in an unpredictable state. Thus, all branches, jumps, traps, Load Program Status, Supervisor Call, and Wait instructions cannot be subject instructions. Software is responsible for insuring that these instructions do not occur as the subject of a branch with execute instruction. No hardware is provided to detect these illegal branch with execute subject instructions. "Illegal Branch With Execute Subject Instructions" on page 183 lists the illegal branch with execute subject instructions.

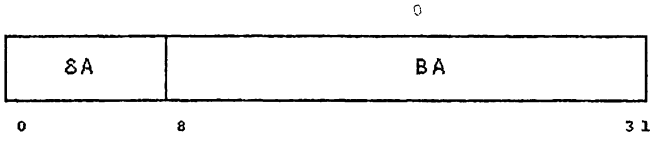
Also, note that, in the case of branch and link with execute instructions, the register containing the return address is available to the subject instruction; hence the subject instruction must be constructed so as not to modify the return address unintentionally. Finally, if the subject instruction is a Move From SCR, where the SCR is the IAR, (move from SCR 13), the results of the move are unpredictable.

All branch and jump instructions are non-privileged.

3.4.1 Branch And Link Instructions

Branch and Link Absolute BA Format

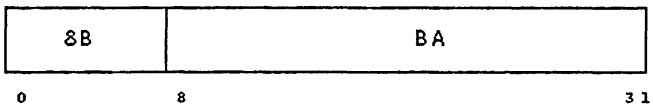
```
BALA    BA
```



The content of register 15 is replaced by the updated instruction address, and the updated instruction address is replaced by the field BA, with its rightmost bit forced to zero.

Branch and Link Absolute with Execute BA Format

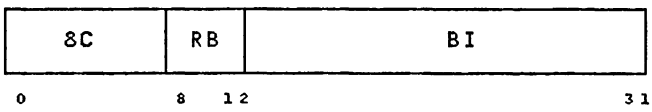
```
BALAX   BA
```



The content of register 15 is replaced by the updated instruction address incremented by four, and the updated instruction address is replaced by the field BA with its rightmost bit forced to zero. The instruction immediately following the branch instruction is executed before the target instruction is executed.

Branch and Link Immediate BI Format

```
BALI    RB,BI
```

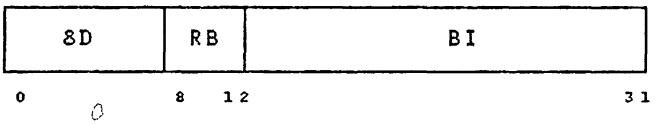


The content of register RB is replaced by the updated instruction address. The updated instruction address is replaced by the field

BI, sign extended and shifted left one bit, added to the branch instruction address.

Branch and Link Immediate with Execute BI Format

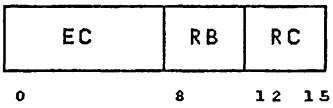
BALIX RB,BI



The content of register RB is replaced by the updated instruction address incremented by four. The updated instruction address is replaced by the field BI, sign extended and shifted left one bit, added to the branch instruction address. The instruction immediately following the branch instruction is executed before the target instruction is executed.

Branch and Link R Format

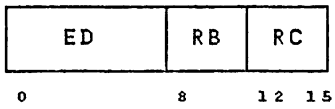
BALR RB, RC



The content of register RB is replaced by the updated instruction address. The updated instruction address is replaced by the content of register RC with the rightmost bit forced to zero.

Branch and Link with Execute R Format

BALRX RB, RC

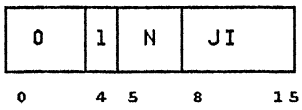


The content of register RB is replaced by the updated instruction address incremented by four, and the updated instruction address is replaced by the content of register RC with the rightmost bit forced to zero. The instruction immediately following the branch instruction is executed before the target instruction is executed.

3.4.2 Conditional Branches

Jump on Condition Bit JI Format

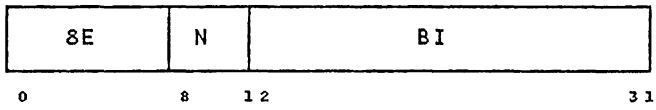
JB N,JI



If the Condition Status bit specified by N is one, the updated instruction address is replaced by the field JI, sign extended and shifted left one bit, added to the branch instruction address. If the specified Condition Status bit is zero, the updated instruction address is unaltered. The field N references only the rightmost eight bits of the Condition Status (bits 24-31).

Branch on Condition Bit Immediate BI Format

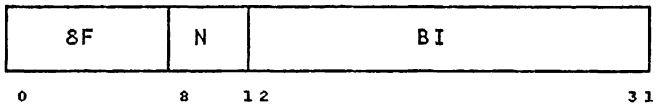
BB N, BI



If the Condition Status bit specified by N is one, the updated instruction address is replaced by the field BI, sign extended and shifted left one bit, added to the branch instruction address. If the specified Condition Status bit is zero, the updated instruction address is unaltered.

Branch on Condition Bit Immediate with Execute BI Format

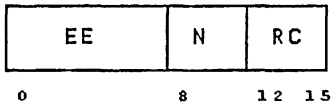
BBX N,BI



If the Condition Status bit specified by N is one, the updated instruction address is replaced by the field BI, sign extended and shifted left one bit, added to the branch instruction address. The instruction immediately following the branch instruction is executed before the target instruction is executed. If the specified Condition Status bit is zero, the updated instruction address is unaltered, and the subject instruction is executed in a normal manner.

Branch on Condition Bit R Format

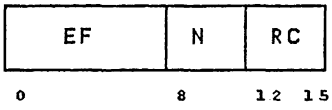
BBR N,RC



If the Condition Status bit specified by N is one, the updated instruction address is replaced by the content of register RC with the rightmost bit forced to zero. If the specified Condition Status bit is zero, the updated instruction address is unaltered.

Branch on Condition Bit with Execute R Format

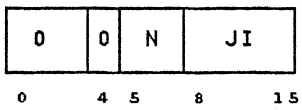
BBRX N,RC



If the Condition Status bit specified by N is one, the updated instruction address is replaced by the content of register RC with the rightmost bit forced to zero. The instruction immediately following the branch instruction is executed before the target instruction is executed. If the specified Condition Status bit is zero, the updated instruction address is unaltered, and the subject instruction is executed in a normal manner.

Jump on Not Condition Bit JI Format

JNB N, JI

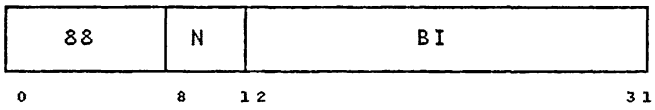


If the Condition Status bit specified by N is zero, the updated instruction address is replaced by the field JI, sign extended and shifted left one bit, added to the branch instruction address. If

the specified Condition Status bit is one, the updated instruction address is unaltered. The field N references only the rightmost eight bits of the Condition Status (bits 24-31).

Branch on Not Condition Bit Immediate BI Format

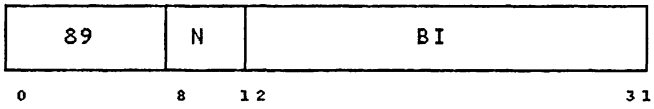
BNB N, BI



If the Condition Status bit specified by N is zero. the updated instruction address is replaced by the field BI, sign extended and shifted left one bit, added to the branch instruction address. If the specified Condition Status bit is one, the updated instruction address is unaltered.

Branch on Not Condition Bit Immediate with Execute BI Format

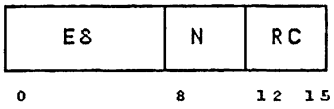
BNBX N, BI



If the Condition Status bit specified by N is zero, the updated instruction address is replaced by the field BI, sign extended and shifted left one bit, added to the branch instruction address. The instruction immediately following the branch instruction is executed before the target instruction is executed. If the specified Condition Status bit is one, the updated instruction address is unaltered, and the subject instruction is executed in a normal manner.

Branch on Not Condition Bit R Format

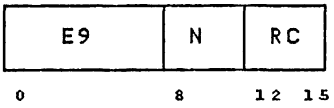
BNBR N, RC



If the Condition Status bit specified by N is Zero, The updated instruction address is replaced by the content of register RC with the rightmost bit forced to zero. If the specified Condition Status bit is one, the updated instruction address is unaltered.

Branch on Not Condition Bit with Execute R Format

BNBRX N, RC



If the Condition Status bit specified by N is zero, the updated instruction address is replaced by the content of register RC with the rightmost bit forced to zero. The instruction immediately following the branch instruction is executed before the target instruction is executed. If the specified Condition Status bit is one, the updated instruction address is unaltered, and the subject instruction is executed in a normal manner.

3.5 TRAPS

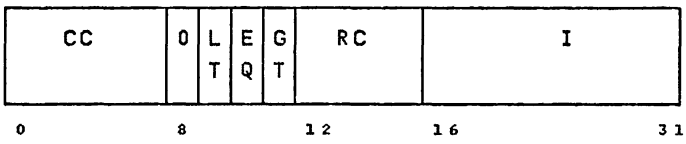
The trap instructions are provided to test for a specified set of conditions. If the conditions tested by a trap instruction are met, the program trap bit of the Program Check Status is set to one and a program check occurs. If the tested conditions are not met, instruction execution continues with the next sequential instruction.

The comparisons are preformed on operands which are treated as 32-bit unsigned integers (logical quantities). The Condition Status is not changed by any of these instructions.

All trap instructions are non-privileged.

Trap On Condition Immediate D Format

TI COND,RC,I



If any of the trap conditions specified by bits 9-11 are met by comparing the content of register RC with the value of the sign extended I-field, the trap bit of the PC register is set, and a program check occurs.

Trap conditions are selected by bits 9-11 as defined below.

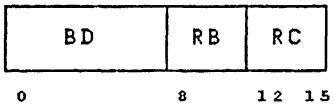
Bit 9 Trap if register RC is less than the value of the sign extended I-field. The trap is enabled if the bit is one and disabled if zero.

Bit 10 Trap if register RC is equal to the value of the sign extended I-field. The trap is enabled if the bit is one and disabled if zero.

Bit 11 Trap if register RC is greater than the value of the sign extended I-field. The trap is enabled if the bit is a one and disabled if zero.

Trap if Register Greater Than or Equal R Format

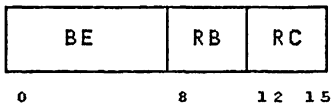
```
TGTE   RB, RC
```



If the content of register RB is greater than or equal to the content of register RC, the Trap bit of the PCS is set, and a program check occurs.

Trap if Register Less Than R Format

```
TLT    RB, RC
```



If the content of register RB is less than the content of register RC, the Trap bit of the PCS is set, and a program check occurs.

3.6 MOVES AND INSERTS

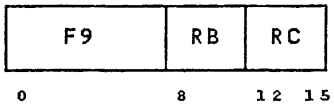
This group of instructions is concerned with the movement of data between general-purpose registers, and between a general-purpose register and the Test Bit of the Condition Status. Except when data is moved into the Test Bit, none of these instructions alter the Condition Status.

All move and insert instructions are non-privileged.

3.6.1 Move Character Instructions

Move Character Zero From Three R Format

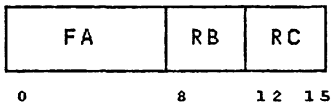
MC03 RB, RC



Character C0 of register RB is replaced by character C3 of register RC.

Move Character One From Three R Format

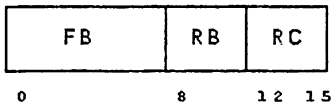
MC13 RB, RC



Character C1 if register RB is replaced by character C3 of the register RC.

Move Character Two From Three R Format

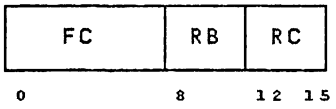
MC23 RB, RC



Character C2 of register RB is replaced by character C3 of register RC.

Move Character Three From Three R Format

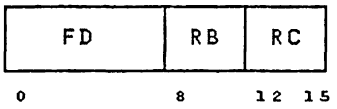
MC33 RB, RC



Character C3 of register RB is replaced by character C3 of register RC.

Move Character Three From Zero R Format

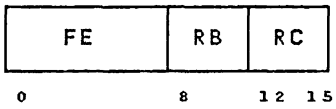
MC30 RB, RC



Character C3 of register RB is replaced by character C0 of register RC.

Move Character Three From One R Format

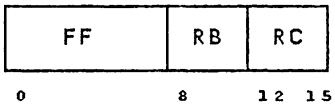
MC31 RB, RC



Character C3 of register RB is replaced by character C1 of register RC.

Move Character Three From Two R Format

MC32 RB, RC

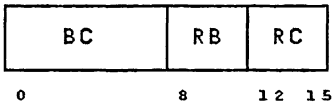


Character C3 of register RB is replaced by character C2 of register RC.

3.6.2 Move To And From Test Bit Instructions

Move From Test Bit R Format

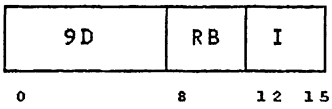
MFTB RB, RC



The bit of register RB specified by the value of bits 27-31 of register RC is set to the value of the Condition Status Test Bit.

Move From Test Bit Immediate Lower Half R Format

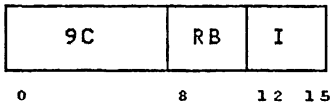
MFTBIL RB, I



The bit of the lower half of register RB specified by I is set to the value of the Condition Status Test Bit.

Move From Test Bit Immediate Upper Half R Format

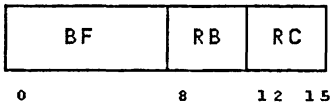
MFTBIU RB, I



The bit of the upper half of register RB specified by I is set to the value of the Condition Status Test Bit.

Move to Test Bit R Format

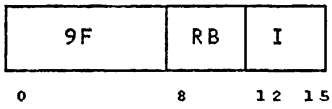
MTTB RB, RC



The Condition Status Test Bit is set to the value of the bit of register RB specified by the value of bits 27-31 of register RC.

Move to Test Bit Immediate Lower Half R Format

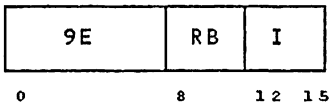
MTTBIL RB, I



The Condition Status Test Bit is set to the value of the bit in the lower half of register RB specified by I.

Move To Test Bit Immediate Upper Half R Format

MTTBIU RB, I



The Condition Status Test Bit is set to the value of the bit in the upper half of register RB specified by I.

3.7 ARITHMETIC

The arithmetic operations treat the general purpose registers as 32 bit quantities in two's complement representation. Each of these instructions affects certain bits in the Condition Status field. However, the bits which are set, and the manner in which they are set, may vary according to the instruction which is executed.

The LT bit is set by all instructions except Multiply Step, Divide Step, and compares to indicate the sign of the result. That is, the LT bit is set to one if the sign bit of the result is one. The arithmetic compare instructions set this bit to one if the algebraic magnitude of a given comparand is less than the algebraic magnitude of the other. The logical compare instructions set this bit to one if the unsigned magnitude of a given comparand is less than the unsigned magnitude of the other. The instructions Multiply Step and Divide Step do not affect this bit.

The EQ bit is set by all instructions except Multiply Step and Divide Step if the result is a field of 32 zeroes, or, in the case of the compare instructions, if the two comparands are equal. The Multiply Step and Divide Step instructions do not affect this bit.

The GT bit is set by all instructions except Multiply Step, Divide Step, and compares to indicate the sign of a non-zero result; it is set to one if the sign bit of a non-zero result is zero. The arithmetic compare instructions set this bit if the algebraic magnitude of a given comparand is greater than the algebraic magnitude of the other. The logical compare instructions set this bit to one if the unsigned magnitude of a given comparand is greater than the unsigned magnitude of the other. The instructions Multiply Step and Divide Step do not affect this bit.

The C0 bit in the Condition Status is set by all instructions except compares, Extend Sign, Divide Step, and Multiply Step to reflect the carry out of bit position zero. The Extend Sign instruction does not affect C0, and the Multiply Step and Divide Step instructions set C0 according to certain multiply and divide conditions. Add operations set C0 to one if a carry occurs and to zero if no carry occurs. Subtract operations set C0 to zero if a borrow occurs and to one if no borrow occurs.

The OV bit is set by all instructions except Extend Sign, Multiply Step, and Divide Step, and compares to indicate arithmetic overflow, i.e., it is set to one when the signed result of an operation cannot be represented by 32 bits. The Extend Sign and Multiply Step instructions do not affect this bit, and the Divide Step instruction sets it according to a divide condition.

The extended operations incorporate the state of the C0 bit into the result. The extended add instructions, AE and AEI, cause the

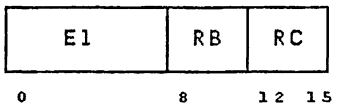
value of the C0 bit to be added to the sum of the two operands. In the extended subtract instruction, SE, the value of the first operand is added to the ones complement of the second operand, and to this result is added the value of C0 bit.

All arithmetic instructions are non-privileged.

3.7.1 Add Instructions

Add R Format

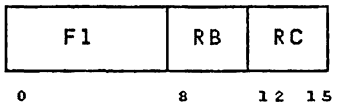
A RB,RC



The contents of registers RB and RC are added and the result placed into register RB. Condition Status bits LT, EQ, GT, C0 and OV are affected.

Add Extended R Format

AE RB,RC



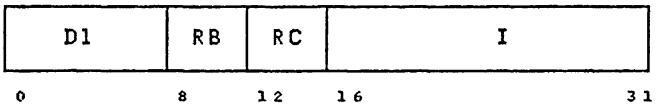
The content of register RB, the Content of register RC, and the value of Condition Status bit C0 are summed and the result placed into register RB. Condition Status bits LT, EQ, GT, C0, and OV are affected.

Programming Note:

This allows multiple precision addition.

Add Extend Immediate D Format

AEI RB,RC,I



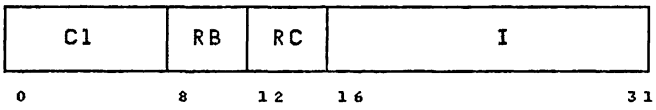
The field I, sign extended, the content of register RC, and the value of Condition Status bit C0 are summed and the result placed in register RB. Condition Status bits LT, EQ, GT, C0, and OV are affected.

Programming Note:

This allows multiple precision addition.

Add Immediate D Format

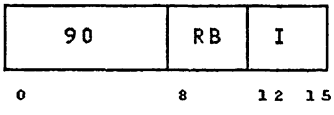
AI RB,RC,I



The field I, sign extended, is added to the content of register RC and the result placed in register RB. Condition Status bits LT, EQ, GT, C0, and OV are affected.

Add Immediate Short R Format

AIS RB,I



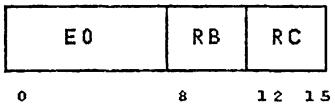
The field I, extended on the left with twenty-eight zeroes is added to the content of register RB and the result placed in

register RB. Condition Status bits LT, EQ, GT, C0 and OV are affected.

3.7.2 Absolute Instruction

Absolute R Format

```
ABS   RB,RC
```

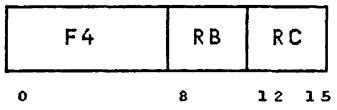


The content of register RB is replaced by the absolute value of the content of register RC. Condition Status bits LT, EQ, GT, C0 and OV are affected. Normally, only Condition Status bits EQ or GT are set to one according to the result. If register RC contains the maximum negative number, for which there is no equivalent positive number, then the content of register RB is set equal to the content of register RC, and the Condition Status bits LT and OV are set to one.

3.7.3 Complement Instructions

One's Complement R Format

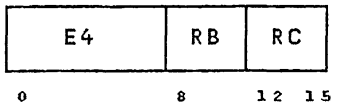
```
ONEC   RB,RC
```



The content of register RB are replaced by the one's complement of the content of register RC. Condition Status bits LT, EQ, and GT are affected.

Two's Complement R Format

TWOC RB,RC

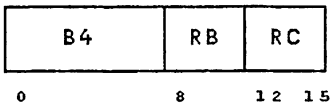


The content of register RB are replaced by the two's complement of the content of register RC. Condition Status bits LT, EQ, GT, C0, and OV are affected.

3.7.4 Compare Instructions

Compare R Format

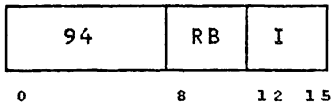
C RB,RC



The contents of registers RB and RC, both treated as 32-bit signed algebraic quantities, are compared. Condition Status bits LT, EQ and GT are affected. Condition Status bits LT and GT are set according to the true relative algebraic magnitudes of the contents of registers RB and RC; that is, LT is set if the content of register RB is algebraically less than the content of register RC, and GT is set if the content of register RB is algebraically greater than the content of register RC. Condition Status bit EQ is set if the content of register RB equals the content of register RC.

Compare Immediate Short R Format

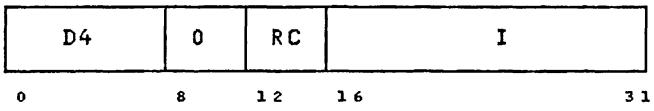
CIS RB,I



The content of register RB is compared to the field I extended on the left with twenty-eight zeroes. Condition Status bits LT, EQ, and GT are affected. Condition Status bits LT and GT are set according to the true algebraic magnitudes of register RB and field I. The LT bit is set if the content of register RB is algebraically less than the field I extended on the left with twenty-eight zeroes, and the GT bit is set if the content of register RB is greater than the field I extended on the left with twenty-eight zeroes. The EQ bit is set if the content of register RB equals the field I extended on the left with twenty-eight zeroes.

Compare Immediate D Format

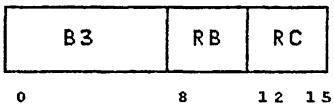
CI RC,I



The content of register RC is compared to field I, sign extended. Condition Status bits LT, EQ, and GT are affected. Condition Status bits LT and GT are set according to the true relative algebraic magnitudes of register RC and field I. The LT bit is set if the content register RC is algebraically less than the field I sign extended, and the GT bit is set if the content of register RC is greater than the field I sign extended. The EQ bit is set if the content of register RC equals the field I, sign extended.

Compare Logical R Format

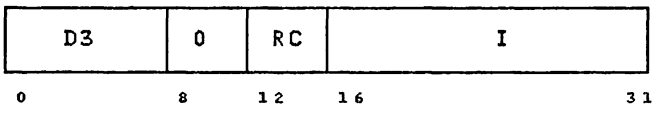
CL RB,RC



The contents of registers RB and RC, both treated as 32-bit unsigned quantities, are compared. Condition Status bits LT, EQ and GT are affected. Condition Status bits LT and GT are set according to the relative unsigned magnitudes of the contents of registers RB and RC; that is, LT is set if the content of register RB is logically less than the content of register RC, and GT is set if the content of register RB is logically greater than the content of register RC. Condition Status bit EQ is set if the content of register RB equals the content of register RC.

Compare Logical Immediate D Format

CLI RC,I

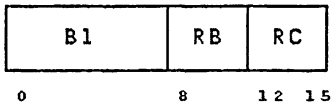


The content of register RC is compared to the field I, sign extended. Condition Status bits LT, EQ and GT are affected. Condition Status bits LT and GT are set according to the relative unsigned magnitudes of register RC and field I sign extended. The LT bit is set if the content of register RC is logically less than the field I sign extended and the GT bit is set if the register RC is greater than the field I sign extended. The EQ bit is set if the content of register RC equals the field I sign extended.

3.7.5 Extend Sign Instruction

Extend Sign R Format

```
EXTS   RB,RC
```

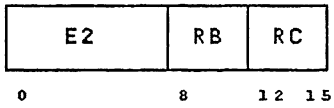


The content of the lower half of register RB is replaced by the lower half of register RC. Bits 0-15 of register RB are set equal to bit 16. Condition Status bits LT, EQ and GT are affected.

3.7.6 Subtract Instructions

Subtract R Format

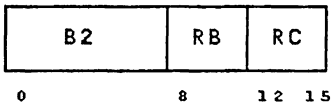
```
S   RB,RC
```



The content of register RC is subtracted from the content of register RB and the result placed into register RB. Condition Status bits LT, EQ, GT, C0 and OV are affected.

Subtract From R Format

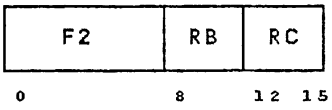
```
SF   RB,RC
```



The content of register RB is subtracted from the content of register RC and the result placed in register RB. Condition Status bits LT, EQ, GT, C0 and OV are affected.

Subtract Extended R Format

SE RB,RC



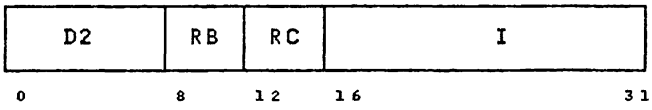
The one's complement of the content of register RC is added to the content of register RB, to which result is added the value of Condition Status bit C0. The result is placed in register RB. Condition Status bits LT, EQ, GT, C0 and OV are affected.

Programming Note:

This allows multiple precision subtraction.

Subtract From Immediate D Format

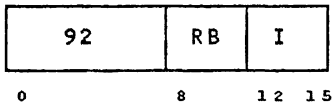
SFI RB,RC,I



The content of register RB is replaced by the content of register RC subtracted from the field I, sign extended. The Condition Status bits LT, EQ, GT, C0 and OV are affected.

Subtract Immediate Short R Format

SIS RB,I

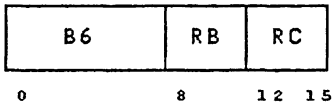


The content of register RB is replaced by the field I subtracted from the content of register RB. For the subtraction, field I is extended on the left with 28 zeroes. Condition Status bits LT, EQ, GT, C0, and OV are affected.

3.7.7 Divide And Multiply Step Instructions

Divide Step R Format

D RB,RC



The content of register RC is added to or subtracted from (RB)//(bit 0 of MQ) depending on whether the signs of registers RB and RC disagree or agree. The 32 rightmost bits of the sum replace the content of register RB. The MQ is shifted left one position and bit 31 of the MQ is set to one if and only if the sign of the 33-bit result equals the sign of register RC. Condition Status bit C0 is set to one if the sign of the 33 bit result equals the sign of the content of register RC, and bit OV is set to one if the sign of the 33-bit result equals the sign of the content of register RB.

Programming Note: Divide Step

The Divide Step instruction can be used to construct algorithms for dividing one number by another. The following example describes an algorithm for dividing a 32-bit dividend by a 32-bit divisor. The operands are in two's complement representation.

Example: Divide X by Y giving quotient Q and remainder R where X, Y, Q and R are 32-bit numbers and Y is not equal to zero, plus one, or minus one.

Initial Conditions: Set general-purpose register RB to the propagated sign of X (zero if X is non-negative, minus one if X is negative). This can be accomplished by loading RB with X and executing a SARI16 RB, 15 instruction. Load Y into general-purpose register RC. Load X into MQ.

Algorithm: Issue the Divide Step instruction with operands RB and RC thirty-two times. If at this point the signs of RB and RC differ, add the content of RC to the content of RB replacing the content of RB. After this test and possible modification of RB, RB contains the preliminary remainder. The MQ contains the 32 rightmost bits of the preliminary quotient. The final quotient and remainder are either equal to the preliminary quotient and remainder or are found by adding one to the preliminary quotient and subtracting the divisor, RC, from the preliminary remainder.
Proof: The Divide Step instruction supports a non-restoring division algorithm. Division is accomplished by repetitive subtraction. The first time, only the sign of the dividend extended to an appropriate width participates in the subtraction. On each subsequent repetition an additional dividend bit is included to the right of the result of the previous repetition; this has the effect of halving the divisor.

Because the division involves binary numbers, the divisor can be subtracted from the current minuend either one or zero times. If it is one, the appropriate quotient bit is one. If it is zero, the quotient bit is zero; however the subtraction has already been performed. Instead of adding the divisor back at this point, half the divisor is added at the next repetition, since the result of subtracting the divisor and adding half the divisor is the same as subtracting half the divisor.

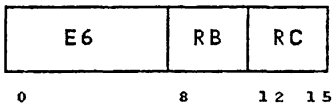
When all dividend bits have been used, if the signs of the divisor and remainder differ, restoration must be done for the last step, and so the divisor is added back into the remainder without changing the quotient. This produces the preliminary quotient and remainder.

The actual algorithm depends on the signs of the divisor and the dividend at each step. These signs also determine whether the initial step is addition or subtraction.

The above algorithm can be modified for dividing a 64-bit dividend contained in RB//MQ. If the initial value of (RB)//(bit 0 of MQ) exceeds the divisor in magnitude, then divide overflow will occur. This condition can be determined by testing for OV=1 after execution of the first Divide Step instruction.

Multiply Step R Format

M RB,RC



The incomplete product of the content of register RC and bits 30 and 31 of the MQ register are formed in (RB)//MQ. A 34-bit sum is formed in accordance with the table below. The MQ is algebraically shifted right two positions with the two rightmost bits of the sum replacing bits 0 and 1 of the MQ. The content of register RB is replaced by the 32 leftmost bits of the sum. Condition Status bit C0 is set to the complement of bit 30 of the MQ before the shift.

Condition Status Bit C0	MQ Bit 30	MQ Bit 31	Algebraic Sum
0	0	0	(RB) + (RC)
0	0	1	(RB) + 2*(RC)
0	1	0	(RB) - (RC)
0	1	1	(RB) + 0
1	0	0	(RB) + 0
1	0	1	(RB) + (RC)
1	1	0	(RB) - 2*(RC)
1	1	1	(RB) - (RC)

Programming Note: Multiply Step

The Multiply Step instructions can be used to construct algorithms for multiplying two numbers together. The following example describes an algorithm for multiplying a 32-bit multiplicand by a 16-bit multiplier. The operands are in two's complement representation.

Example: Multiply X by Y giving Z where X is a 32-bit number and Y is a 16-bit number.

Initial Conditions: Load X into general-purpose register RC; load Y into the MQ, set the content of general-purpose register RB to zero; set Condition Status bit C0 to one. RB and C0 can be initialized simultaneously by executing a S RB,RB instruction.

Algorithm: Issue the Multiply Step instruction with operands RB and RC eight times.

Result: The 16 rightmost bits of the product Z are in the MQ; the 32 leftmost bits are in register RB.

Proof: The 16-bit multiplier Y can be expressed as the sum of 16 terms of the form:

$$y_i 2^i \quad \text{where } y \text{ equals 0 or 1 and } i = 0,1,2...15$$

or eight terms of the form:

$$(y_{2i} + 2*y_{2i+1}) * 4^i$$

where y_{2i} and y_{2i+1} equal 0 or 1 and i = 0,1...,7.

The Multiply Step instruction accumulates a partial sum in register RB and the leftmost bits of the MQ; a Condition Status bit C0 equal to zero indicates a carry. The instruction provides four cases for the parenthesized factor when there is no carry into the term and four cases when there is a carry, as follows:

y 2i	+ 2*y 2i+1	Carry	Value
0		No	(RB)
1		No	(RB)+(RC)
2		No	(RB)+4*(RC)-2*(RC)
3		No	(RB)+4*(RC)-(RC)
0		Yes	(RB)+(RC)
1		Yes	(RB)+2*(RC)
2		Yes	(RB)+4*(RC)-(RC)
3		Yes	(RB)+4*(RC)

In the parenthesized factor, y2i is the value of MQ bit 31 and y2i+1 is the value of MQ bit 30. Whenever MQ bit 30 is one, the term 4*(RC) appears. This is a carry into the next partial sum.

The Multiply Step instruction places the rightmost two bits of the partial sum in vacated MQ bits (bits 0 and 1). This provides the 4i factor since it has the effect of multiplying the remaining bits of the multiplier by four.

3.8 LOGICAL OPERATIONS

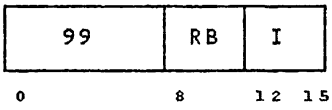
The logical operations treat the contents of the general-purpose registers as 32-bit unsigned integers with the exception of the instruction Count Leading Zeroes (CLZ), which is applied to the lower half of a register. All logical operations except CLZ set Condition Status bits LT, EQ and GT according to the algebraic value expressed in two's complement representation. If the result is a negative value, LT is set to one; if it is zero, EQ is set to one; if it is positive and not zero, GT is set to one. The Condition Status is unaffected by the Count Leading Zeroes instruction.

All logical instructions are non-privileged.

3.8.1 Clear And Set Bit Instructions

Clear Bit Lower Half R Format

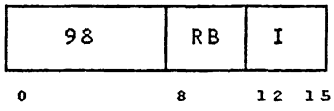
```
CLRBL   RB,I
```



A bit in the lower half of register RB is set to zero, where the bit is specified by the immediate field I. Condition Status bits LT, EQ, and GT are affected.

Clear Bit Upper Half R Format

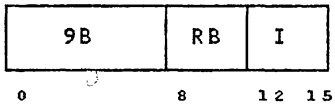
```
CLRBU   RB,I
```



A bit in the upper half of register RB is set to zero, where the bit is specified by the immediate field I. Condition Status bits LT, EQ, and GT are affected.

Set Bit Lower Half R Format

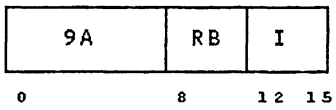
SETBL RB,I



A bit in the lower half of register RB is set to one, where the bit is specified by the immediate field I. Condition Status bits LT, EQ and GT are affected.

Set Bit Upper Half R Format

SETBU RB,I

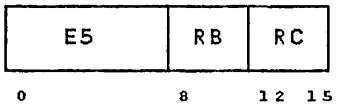


A bit in the upper half of register RB is set to one, where the bit is specified by the immediate field I. Condition Status bits LT, EQ and GT are affected.

3.8.2 AND Instructions

AND R Format

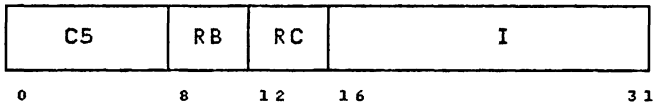
```
N     RB,RC
```



The AND of the contents of registers RB and RC replaces the content of the register specified by RB. Condition Status bits LT, EQ and GT are affected.

AND Immediate Lower Half Extended Zeroes D Format

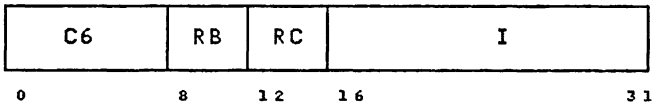
```
NILZ   RB,RC,I
```



The AND of the I field extended on the left with sixteen zeroes and the content of register RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

AND Immediate Lower Half Extended Ones D Format

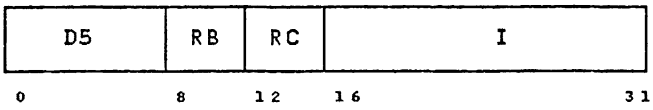
```
NILO   RB,RC,I
```



The AND of the I field extended on the left with sixteen ones and the content of register RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

AND Immediate Upper Half Extended Zeroes D Format

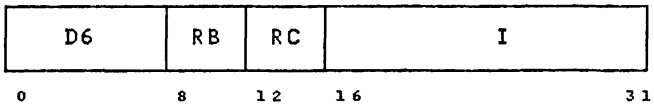
NIUZ RB,RC,I



The AND of the I field extended on the right with sixteen zeroes and the content of register RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

AND Immediate Upper Half Extended Ones D Format

NIUO RB,RC,I

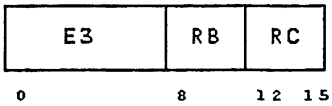


The AND of the I field extended on the right with sixteen ones and the content of register RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

3.8.3 OR Instructions

OR R Format

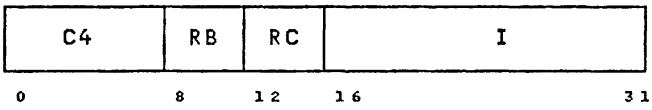
O RB,RC



The OR of the contents of registers RB and RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

OR Immediate Lower D Format

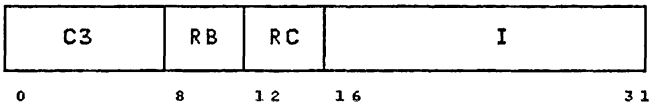
OIL RB,RC,I



The OR of the I field extended on the left with sixteen zeroes and the content of register RC replaces the content of register RB. Condition Status bit s LT, EQ and GT are affected.

OR Immediate Upper D Format

OIU RB,RC,I

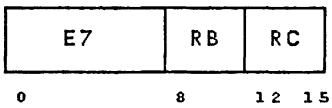


The OR of the I field extended on the right with sixteen zeroes and the content of register RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

3.8.4 Exclusive OR Instructions

Exclusive OR R Format

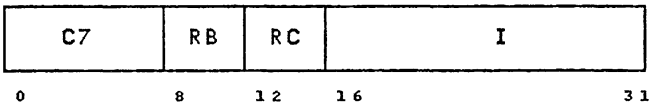
X RB, RC



The EXCLUSIVE OR of the contents of registers RB and RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

Exclusive OR Immediate Lower Half D Format

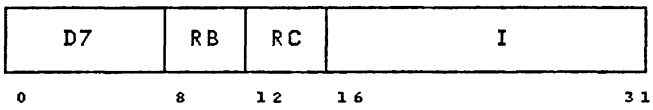
XIL RB,RC,I



The EXCLUSIVE OR of the I field extended on the left with sixteen zeroes and the content of register RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

Exclusive OR Immediate Upper Half D Format

XIU RB,RC,I

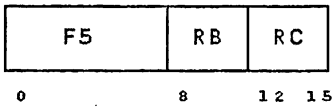


The EXCLUSIVE OR of the I field extended on the right with sixteen zeroes and the content of register RC replaces the content of register RB. Condition Status bits LT, EQ and GT are affected.

3.8.5 Count Leading Zeroes Instruction

Count Leading Zeroes R Format

```
CLZ       RB,RC
```



The content of register RB is replaced by the binary representation of the number of leading zeroes in the lower half of register RC (i.e., the number of zeroes to the left of the leftmost one bit in the lower half of register RC).

Programming Note:

If the lower half of register RC is equal to zero, the content of register RB is replaced by the binary representation of sixteen.

3.9 SHIFTS

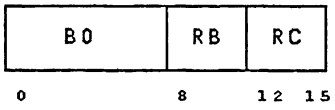
Shift instructions operate on either the content of a register or a register half. Immediate form shifts specify a shift amount of 0 to 31 bits to the left or right based on the value of the immediate field. Indirect shifts specify a shift amount of 0 to 63 bits to the left or right based on the low-order six bits of register RC. A shift amount greater than 31 bits results in a 32-bit shift. All shifts set the Condition Status bits LT, EQ and GT according to the resultant algebraic value returned to the register. All instructions except the shift algebraic right instructions supply zeroes to the vacated bit positions.

All shift instructions are non-privileged.

3.9.1 Shift Algebraic Right Instructions

Shift Algebraic Right R Format

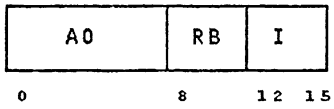
SAR RB,RC



The content of register RB is shifted right the number of bit positions specified by bits 26-31 of register RC. Bits equal to the original sign bit (bit 0) are supplied to the vacated high-order positions. Condition Status bits LT, EQ and GT are affected.

Shift Algebraic Right Immediate R Format

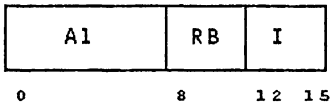
SARI RB,I



The content of register RB is shifted right the number of bit positions specified by the field I. Bits equal to the original sign bit (bit 0) are supplied to the vacated high-order positions. Condition Status bits LT, EQ and GT are affected.

Shift Algebraic Right Immediate Plus Sixteen R Format

SARI16 RB,I

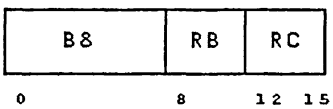


The content of the register RB is shifted right the number of bit positions specified by the field I plus sixteen. Bits equal to the original sign bit (bit 0) are supplied to the vacated high-order positions. Condition Status bits LT, EQ and GT are affected.

3.9.2 Shift Right Instructions

Shift Right R Format

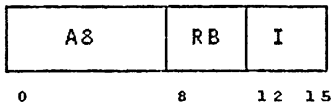
SR RB,RC



The content of register RB is shifted right the number of bit positions specified by bits 26-31 of register RC. Zeroes are supplied to the vacated high-order positions. Condition Status bits LT, EQ and GT are affected.

Shift Right Immediate R Format

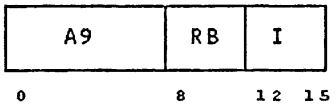
SRI RB,I



The content of register RB is shifted right the number of bit positions specified by the field I. Zeroes are supplied to the vacated high-order positions. Condition Status bits LT, EQ and GT are affected.

Shift Right Immediate Plus Sixteen R Format

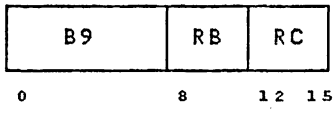
SRI16 RB,I



The content of register RB is shifted right the number of bit positions specified by the field I plus sixteen. Zeroes are supplied to the vacated high-order positions. Condition Status bits LT, EQ and GT are affected.

Shift Right Paired R Format

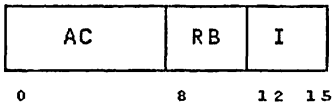
SRP RB,RC



The content of the register RB shifted right the number of bit positions specified by bits 26-31 of register RC with zeroes supplied to the vacated high-order positions is placed in the twin register RB. The content of register RB is not affected. Condition Status bits LT, EQ and GT are affected.

Shift Right Paired Immediate R Format

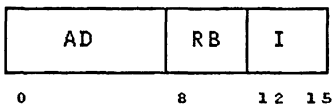
SRPI RB,I



The content of register RB shifted right the number of bit positions specified by the field I with zeroes supplied to the vacated high-order positions is placed in the twin register RB. The content of register RB is not affected. Condition Status bits LT, EQ and GT are affected.

Shift Right Paired Immediate Plus Sixteen R Format

SRPI16 RB,I



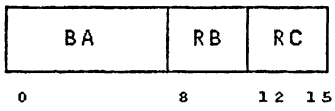
The content of register RB shifted right the number of bit positions specified by the field I plus sixteen with zeroes supplied to the vacated high-order positions is placed in the twin

of register RB. The content of register RB is not affected. Condition Status bits LT, EQ and GT are affected.

3.9.3 Shift Left Instructions

Shift Left R Format

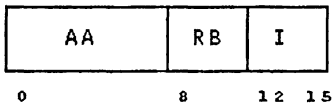
SL RB,RC



The content of register RB is shifted left the number of bit positions specified by bits 26-31 of register RC. Zeroes are supplied to the vacated low-order positions. Condition Status bits LT, EQ and GT are affected.

Shift Left Immediate R Format

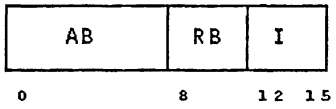
SLI RB,I



The content of the register RB is shifted left the number of bit positions specified by the field I. Zeroes are supplied to the vacated low-order positions. Condition Status bits LT, EQ and GT are affected.

Shift Left Immediate Plus Sixteen R Format

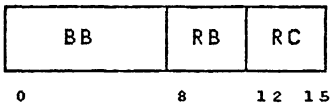
SLI16 RB,I



The content of register RB is shifted left the number of bit positions specified by the field I plus sixteen. Zeroes are supplied to the vacated low-order positions. Condition Status bits LT, EQ and GT are affected.

Shift Left Paired R Format

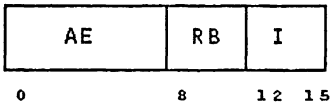
SLP RB,RC



The content of register RB shifted left the number of bit positions specified by bits 26-31 of register RC with zeroes supplied to the vacated low order positions is placed in the twin of register RB. The content of register RB is not affected. Condition Status bits LT, EQ and GT are affected.

Shift Left Paired Immediate R Format

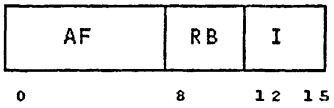
SLPI RB,I



The content of register RB shifted left the number of bit positions specified by the field I with zeroes supplied to the vacated low order positions is placed in the twin of register RB. The content of register RB is not affected. Condition Status bits LT, EQ and GT are affected.

Shift Left Paired Immediate Plus Sixteen R Format

SLPI16 RB,I



The content of the register RB shifted left the number of bit positions specified by the field I plus sixteen with zeroes supplied to the vacated low order positions is placed in the twin of register RB. The content of register RB is not affected. Condition Status bits LT, EQ and GT are affected.

3.10 SYSTEM CONTROL

The system control instructions provide a means of examining and manipulating the state of certain processor facilities. This is done through two sub-classes of instructions. The first sub-class operates on the contents of the system control registers. These instructions allow the reading or writing of any SCR or the setting or clearing of any of the low-order 16 bits of the SCR. A second sub-class of instructions within this class provides the necessary software interface to the interrupt facility described in "Interrupts" on page 17.

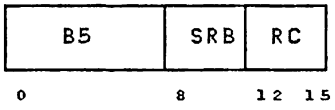
The instructions which deal with the SCRs provide a general capability of operating on any of the SCRs. However, because of the definition of certain SCRs, not every operation on an SCR gives a predictable result. Moreover, bits in any SCR which have been specified as reserved bits cannot be used in a predictable manner. These exceptional cases are specified along with the instruction definitions. Finally, all SCRs except the ICS (SCR 14) are dynamically changed by the processor, often asynchronously to instruction sequencing. Hence, a read of an SCR following a write will not necessarily get the same data which was written.

Only certain system control instructions are non-privileged. The non-privileged instructions are MTS, MFS, SETSB, and CLRSB when the SCR reference by these instructions is the MQ or CS, and the SVC instruction An attempt to execute any other system control instruction in problem state will cause the privileged instruction exception bit in the program check status to be set and a program check to occur. Refer to "Program-Check Errors" on page 124 for a description of the program check status.

3.10.1 Move To And From SCR Instructions

Move to SCR R Format

MTS SRB,RC

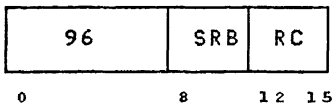


The content of system control register SRB is replaced by the content of register RC. Any reserved bits in the specified SCR

are not set to predictable values. If the specified SCR is the IAR (SCR 13), the results of this instruction are unpredictable.

Move from SCR R Format

MFS SRB,RC

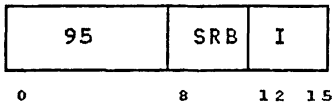


The content of register RC is replaced by the content of system control register SRB. The bits of register RC corresponding to reserved bits of the specified SCR are set to unpredictable values. If the specified SCR is the IAR (SCR 13), the value which is loaded into register RC is the address of the instruction immediately following the MFS instruction in main storage.

3.10.2 Clear And Set SCR Bit Instructions

Clear SCR Bit R Format

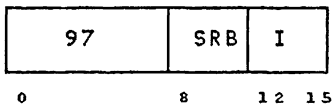
CLRSB SRB,I



A bit in the lower half of system control register SRB is set to zero, where the bit is selected by the immediate field I. If the selected bit of the SCR is a reserved bit, it is not set to a predictable value. If the specified SCR is the IAR (SCR 13), the results of this instruction are unpredictable.

Set SCR Bit R Format

SETSB SRB,I

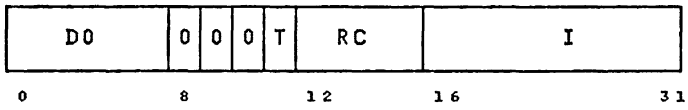


A bit in the lower half of system control register SRB is set to one, where the bit is selected by the immediate field I. If the selected bit of the SCR is a reserved bit, it is not set to a predictable value. If the specified SCR is the IAR (SCR 13). The results of this instruction were unpredictable.

3.10.3 Load Program Status Instruction

Load Program Status D Format

LPS T, I(RC)



The content of the IAR (SCR 13) is replaced by the word in main storage addressed by 0/(RC) plus the sign extended I-field. The content of the ICS (SCR 14) is replaced by the content of the main storage halfword addressed by 0/(RC) plus the sign extended I-field plus four. The content of the CS (SCR 15) is replaced by the content of the main storage halfword addressed by 0/(RC) plus the sign extended I-field plus six. Any reserved bits in the SCRs are set to unpredictable values. If the processor is on the Machine Check level (see "Machine-Check Error Handling" on page 122) when the LPS is executed, the content of the MCS is set to zero. If the processor is on the Program Check level (see "Program-Check Error Handling" on page 124) when the LPS is executed, the content of the PCS is set to zero.

If bit 11 of this instruction is a one, interrupts remain pending until the target instruction of the LPS instruction has been

executed. If bit 11 is zero, interrupts may occur after the LPS instruction is executed.

Programming Note:

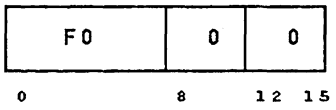
The LPS instructions may be used to return from an interrupt.

The LPS instruction may also be used to trace instruction execution. This is accomplished by setting a bit in the IRB to generate an interrupt request before executing the LPS instruction. The bit which is set should have a corresponding interrupt request priority greater than the processor priority which is loaded by the LPS instruction. If the Interrupt Mask which is loaded by the LPS is zero, and if bit 11 of the LPS instruction is a one, an interrupt will occur after the target instruction of the LPS has been executed.

3.10.4 Wait Instruction

Wait R Format

WAIT

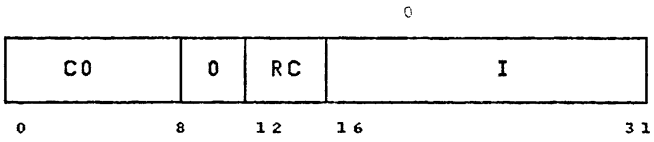


The processor is placed in the wait state. When the processor is in the wait state it does not execute any instructions nor make any storage accesses. The processor is removed from the wait state through the occurrence of an interrupt, error, or power-on reset.

3.10.5 Supervisor Call Instruction

Supervisor Call D Format

SVC I(RC)



The content of the IAR (SCR 13) is stored into the word in main storage beginning at address X'190'. The content of the ICS (SCR 14) is stored into the halfword in main storage beginning at address X'194'. The content of the CS (SCR 15) is stored into the halfword in main storage beginning at address X'196'. The low-order 16-bits of the 32-bit sum 0/(RC) + 0[16]//I is stored into the halfword in main storage beginning at address X'19E'.

The content of the IAR (SCR 13) is replaced by the word in main storage beginning at address X'198'. The content of the ICS (SCR 14) is replaced by the content of the halfword in main storage beginning at address X'19C'. Any reserved bits in the IAR and the ICS are set to unpredictable values.

3.11 INPUT/OUTPUT

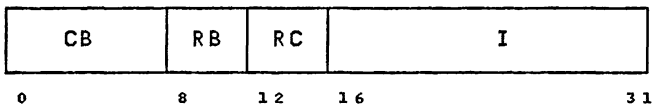
Programmed I/O (PIO) instructions are used to transfer data between the general-purpose registers and system components.

All I/O addresses are considered to be device addresses. The upper byte of the I/O address is checked to be zero. A non-zero high order byte in the I/O address will cause a program check.

All PIO instructions are non-privileged. Each I/O device determines whether it is a privileged or non-privileged device. Privileged I/O devices accept I/O commands from the processor only when the processor is in supervisor state. An attempt to access a privileged I/O device from problem state will cause the Data Address Exception bit in the Program Check Status to be set and a program check to occur. See "Privileged I/O Device Connection" on page 90 for a description of privileged I/O device connection, and "Program-Check Status" on page 125 for a description of the Program Check Status.

Input/Output Read D Format

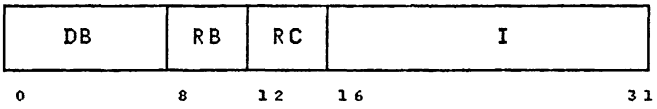
```
IOR     RB,I(RC)
```



The content of register RB is replaced by data transferred from the I/O device selected by the effective address 0/(RC) + 0[16]//I. Bits 8-31 of the 32-bit effective address are interpreted as the I/O device address. Bits 0-7 of the effective address must be zero.

Input/Output Write D Format

IOW RB, I(RC)



The content of register RB is transferred to the I/O device selected by the effective address 0/(RC) + 0[16]//I. Bits 8-31 of the 32-bit effective address are interpreted as the I/O device address. Bits 0-7 of the effective address must be zero.

4.0 INPUT/OUTPUT FACILITY

4.1 I/O CAPABILITY

The ROMP system provides two capabilities for controlling I/O operations: programmed I/O (PIO), and I/O interrupts.

4.1.1 Programmed I/O

Two programmed I/O (PIO) instructions (IOR and IOW) provide I/O operations which are synchronous to the program. For each PIO instruction executed, a 24-bit I/O address field is sent to an I/O device and data is transferred between the I/O Device and a general purpose register. The PIO instructions are defined in "Input/Output" on page 88.

4.1.2 Privileged I/O Device Connection

The ROMP architecture allows the system designer to determine whether each I/O device is privileged or non-privileged. Privileged I/O devices can be accessed only by programs executing in supervisor state. An attempt to access a privileged I/O device from problem state will cause the Data Address Exception bit in the Program Check Status to be set and a program check to occur.

The determination of privileged or non-privileged mode can be made in each device by selectively including the problem state signal from the processor in the decode logic used to accept an I/O command. Each I/O device will normally contain address decode logic which is used to determine if a particular I/O command is directed to the device. A device can be made privileged by including the ROMP Storage Channel (RSC) problem state signal (DAL06) in the address decode logic. This will allow the device to accept I/O commands only if the processor is in supervisor state. An attempt to access the I/O device from problem state will cause the command to not be recognized by the I/O device. If the I/O command is not recognized, no ACK/NAK response will be generated. This will cause the Data Address Exception bit in the Program Check Status to be set and a program check to occur.

4.1.3 I/O Interrupt Requests

I/O interrupt requests report asynchronous events. Each interrupt request is assigned one of seven priority levels. Processor logic allows I/O interrupts (unless masked) on a priority basis. The interrupt facility is described in "Interrupts" on page 17.

5.0 ROMP STORAGE CHANNEL

5.1 GENERAL DESCRIPTION

The ROMP Storage Channel (RSC) is a high-bandwidth synchronous bus designed to interconnect a ROMP, a storage unit, and one or more RSC devices. It supports a 32-bit data transfer and a 24-bit (optionally 32-bit) address. Read operations on the RSC consist of two uncoupled transfers, a request and a reply, which allows multiple operations to overlap. This feature, combined with several features in the ROMP data flow, allows high processor performance with relatively slow storage through interleaving techniques.

The main elements of the RSC are a 32-bit (plus 4 parity) multiplexed Data/Address bus and a 5-bit (plus 1 parity) Tag bus. The Data/Address bus contains either 32-bits of data or a 24-bit address plus a byte of control information. The Tag bus contains codes which link replies to requests. An optional Address Extension bus provides 8 high-order address bits which extend the address to 32 bits. In addition, there are several miscellaneous handshaking, control, and clock lines.

The RSC runs synchronously with ROMP, with two RSC cycles per ROMP cycle. The first RSC cycle is always used to transmit addresses, and the second is used for data. There are three types of RSC transfers:

1. A read request where one device on the RSC is requesting data from another device. A read request consists of a single address cycle. Note that a read request always results in a reply.

2. A write request where one device on the RSC is writing data to another device. A write request consists of an address cycle plus the following data cycle.

3. A reply where one device is sending data to another device that previously requested a read. A reply consists of a single data cycle.

These requests are shown in Figure 8.

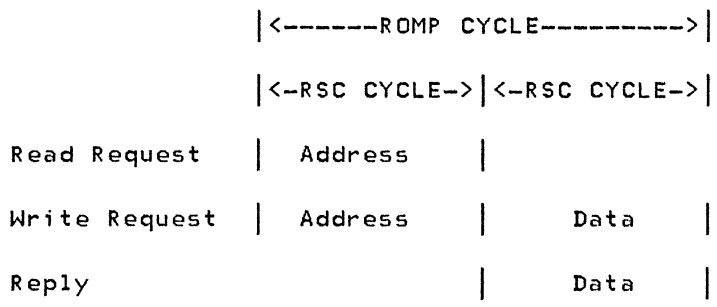


Figure 8. RSC Transfers

The RSC architecture allows any device to assume control of the RSC and issue requests. In a typical system, ROMP would issue requests to storage or RSC devices, and RSC devices would issue requests to storage and each other.

Control of the RSC is determined by two arbitration systems, one for requests (Address Grant) and one for replies (Data Grant). Arbitration is for a period of two RSC cycles, with reply and request arbitration being overlapped in time with each other, and also with bus transfers. The arbitration systems are defined to be daisy-chained, but it is possible to implement a radial arbiter.

A typical RSC system configuration is shown in Figure 9 on page 94.

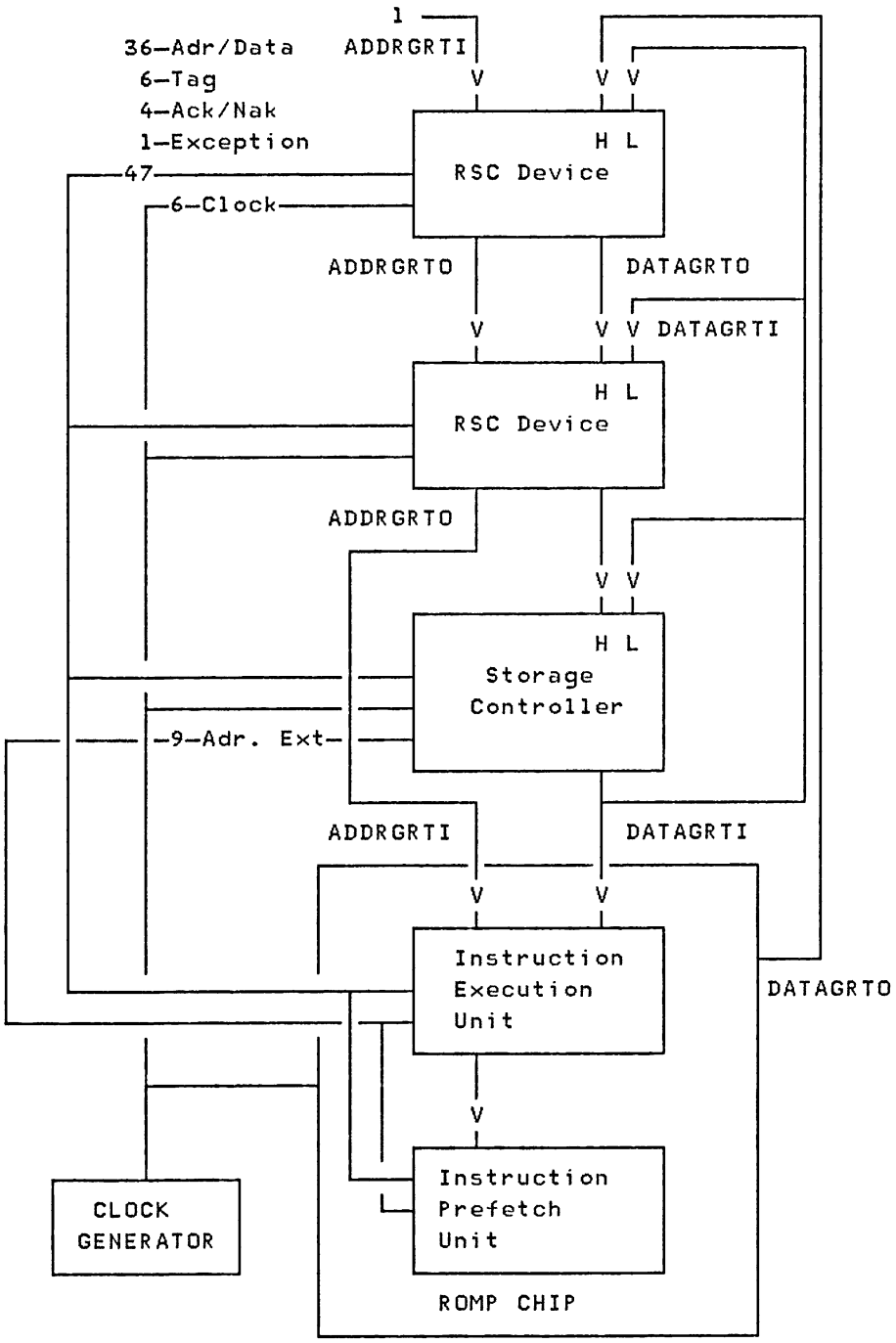


Figure 9. Typical RSC Configuration

5.2 STORAGE CHANNEL DEFINITION

The ROMP Storage Channel (RSC) consists of 57 standard lines, plus 9 address extension lines, divided into five functional groups. This section provides an overview of each of these five functional groups. Subsequent sections provide a detailed description of each functional group.

5.2.1 Address And Data Bus

The first group of signals is the address/data bus. There are 36 bi-directional lines in the bus (32 data bits plus four parity bits). The address/data bus is defined in "Address/Data Bus Definition" on page 97.

5.2.2 Tag Bus

The second group of signals is the tag bus. There are six bi-directional lines in this group (five tag bits plus one tag parity bit). Whenever a request is placed on the RSC, a unique code (a 'tag'), which identifies the source of the request, is placed on the tag bus. This tag is used as a return address for a reply generated in response to the request. The current bus definition uses three tags for special functions (one for channel reset, one for idle condition, and one for write data). The remaining twenty nine are available as return addresses. ROMP uses ten of these, leaving nineteen available for use by RSC devices. The tag bus is defined in "Tag Bus Definition" on page 98.

5.2.3 Control Signals

The third group of signals is the storage channel controls. ACKA, ACKD, NAKA and NAKD are generated in response to transfers on the bus to indicate whether the transfer was successful. ACKA and NAKA are responses to the address transfer and occur during the data cycle which immediately follows the address transfer. ACKD and NAKD are responses to the data transfer cycle and occur during the address cycle which immediately follows the data transfer. These lines are negative true signals, and the processor drives them to the high (inactive) level every other bus cycle. A pullup resistor maintains the inactive level if no system component is attempting to pull the lines low. The four possible combinations of ACK and NAK indicate whether the transfer was successful or whether an error occurred according to the following table.

ACKA or ACKD	NAKA or NAKD	Definition
Inactive	Inactive	No Device Responded
Inactive	Active	Device Busy, Retry Transfer
Active	Inactive	Transfer Successful
Active	Active	Parity Error

Address Grant and Data Grant are groups of signals that are used to arbitrate among devices on the RSC for use of the bus. Address Grant is used to arbitrate for an address cycle and the next data cycle. Data Grant is used to arbitrate for a data cycle. Address Grant and Data Grant are serially connected between devices, starting with the highest priority device and ending with the lowest priority device so that only one device at a time may originate a transfer on the bus. This means that the address grant input (ADDRGRTI) of a given device in the priority chain is connected to the address grant output (ADDRGRTO) of the next higher priority device. Similarly, the data grant input (DATAGRTI) of one device is connected to the data grant output (DATAGRTO) of the next higher priority device. The lowest order DATAGRTO output is sent to all devices in the Address Grant chain, and serves to prevent them from using a data cycle needed for a reply. See "Bus Arbitration" on page 102 for more information.

HOLD is used in systems which have devices on the RSC that can interfere with ROMP access of the RSC. See "Hold Time-Out Counter" on page 108 for more information.

EXCEPTION is used in systems which implement storage protection or address translation. See "Storage Protection and Address Translation" on page 109 for more information.

5.2.4 Address Extension Bus

The fourth group of lines provide 8 bits of address extension plus parity for 32-bit addressing. These lines can be used in virtual address systems where a 32-bit address is desired. Use of these lines is described in "Address Extension Bus Definition" on page 99.

5.2.5 RSC Clocks

The fifth functional group of signals consist of six clocks which control the RSC. See "Storage Channel Clocking" on page 100 for

more information.

5.3 RSC SIGNAL DEFINITIONS

5.3.1 Address/Data Bus Definition

The address/data bus lines (DAL) provide 24 bits of address and 8 bits or control information during an address cycle, and 32 bits of data during a data cycle. An additional eight address bits are available on the Address Extension Bus during an address cycle for systems which use 32-bit addressing.

During an address cycle, the address/data bus is defined as follows:

DAL00	Function
Inactive	Storage Access
Active	Programmed I/O

DAL01	Function
Inactive	Read
Active	Write

DAL02	DAL03	Operand Length
Inactive	Inactive	One Byte
Inactive	Active	Two Bytes
Active	Inactive	Four Bytes
Active	Active	Two Byte Test and Set

DAL04	Function
Inactive	Storage Protection Disabled
Active	Storage Protection Enabled

DAL05	Function
Inactive	Address Translation Disabled
Active	Address Translation Enabled

DAL06	Function
Inactive	Supervisor State
Active	Problem State

DAL07	Reserved
DAL08	Address Bit 0 (MSB)
DAL09	Address Bit 1
DAL10	Address Bit 2
DAL11	Address Bit 3
DAL12	Address Bit 4

DAL13	Address Bit 5
DAL14	Address Bit 6
DAL15	Address Bit 7
DAL16	Address Bit 8
DAL17	Address Bit 9
DAL18	Address Bit 10
DAL19	Address Bit 11
DAL20	Address Bit 12
DAL21	Address Bit 13
DAL22	Address Bit 14
DAL23	Address Bit 15
DAL24	Address Bit 16
DAL25	Address Bit 17
DAL26	Address Bit 18
DAL27	Address Bit 19
DAL28	Address Bit 20
DAL29	Address Bit 21
DAL30	Address Bit 22
DAL31	Address Bit 23 (LSB)

During a data transfer, 32 bits are transferred simultaneously, with DAL00 being the most significant bit and DAL31 being the least significant bit.

5.3.2 Tag Bus Definition

Devices generating requests on the RSC are identified by the code they put on the Tag bus (5 bits plus parity). Whenever a device places a read or write request on the RSC, it places its five-bit code on the Tag bus along with the address on the Address/Data bus to indicate the reply destination. The tag code is saved by the accessed device and placed on the Tag bus with the reply.

The Tag bus is normally used during address cycles to indicate the source of requests, and during data cycles to specify the destination of replies. The Write Data tag is placed on the bus during the data portion of a write request. This tag distinguishes a Write Data parcel from a Reply.

Two other special purpose tags are the Reset tag and the Idle Mode tag. The Reset tag, which is valid only during data cycles, is utilized to reset devices on the RSC. It is placed on the RSC by ROMP after Power On Reset, and in certain error conditions. The Idle Mode tag is used on either the address or data cycle to indicate a channel idle condition. It is placed on the bus whenever a device controls the bus for a cycle but has no valid parcel to transmit.

Two tags (11110 and 11111) are reserved for use by a co-processor and can not be used by other RSC devices. Devices on the RSC

which provide certain protection or checking functions (i.e. storage controllers) treat these tags the same as ROMP tags. This allows a co-processor to have the same access authority as ROMP.

Figure 10 on page 100 summarizes the tag codes. A 0 is an inactive logic level and a 1 is an active logic level. An X indicates a don't care state.

5.3.3 Address Extension Bus Definition

The Address Extension Bus provides an additional 8 bits of address extension, plus parity, for systems using 32-bit addressing. During a request for data by a device, the address extension bus is defined as follows:

Signal Name	Function
ADREXT0	Address Extension Bit 0 (MSB)
ADREXT1	Address Extension Bit 1
ADREXT2	Address Extension Bit 2
ADREXT3	Address Extension Bit 3
ADREXT4	Address Extension Bit 4
ADREXT5	Address Extension Bit 5
ADREXT6	Address Extension Bit 6
ADREXT7	Address Extension Bit 7 (LSB)
ADREXTP	Odd Parity on ADREXT0-ADREXT7

During a data transfer, the address extension bus is defined as follows:

ADREXT0	Function
Inactive	Select 32-bit addressing mode
Active	Select 24-bit addressing mode

ADREXT1	Function
Inactive	Invalid parity on ADREXT0 thru ADREXT7 on previous address cycle
Active	Valid parity on ADREXT0 thru ADREXT7 on previous address cycle

ADREXT2 thru ADREXT7	Reserved
ADREXTP	Reserved

The address extension bus provides an additional 8 address bits for use in virtual address systems where a 32-bit address is desired. These lines are required only in systems that implement 32-bit addressing. If a system implements only 24-bit addressing, the address extension bus signal ADREXT0 and ADREXT1 must be tied

Tag Bits 0 1 2 3 4	Operation
0 0 0 0 0	Idle Mode
0 0 0 0 1	Reset (See "Reset" on page 107)
0 0 0 1 0	Write Data
0 0 0 1 1	Available for Other Devices
0 0 1 0 X	Available for Other Devices
0 0 1 1 X	Processor Data Read/Write
0 1 X X X	Processor Instruction Fetch
1 0 0 0 0 thru 1 1 1 0 1	Available For Other Devices
1 1 1 1 X	Reserved For Co-Processor

Figure 10. Tag Definition

active through a pullup resistor. ADREXT0 is used to enable checking of the upper address byte to insure that it is zero. A non-zero high-order address byte will cause a program check as described in "Storage Access" on page 30. ADREXT1 is used to report parity errors on the address extension bus during the previous address cycle.

Systems which implement 32-bit addressing will drive ADREXT0 inactive during data cycles, to indicate 32-bit addressing is being used. These systems must also check parity on the address extension bus, and use ADREXT1 during data cycles to report parity checking results from the previous address cycle. This parity error signal is ORed with the RSC NAKA signal by the processor to determine if any parity errors occurred during an address cycle transfer.

The address extension bus provides extension of storage addresses to 32-bits. All I/O addresses remain 24-bits. The upper-byte of the 32-bit I/O address is checked to insure that it is zero. A non-zero high-order I/O address byte will cause a program check in both 24-bit and 32-bit addressing mode.

5.3.4 Storage Channel Clocking

ROMP operates with four clocks (-T0, -T1, -T2, -T3) which are generated external to the processor chip. ROMP, storage, and any other devices on the storage channel must use these clocks to control channel transfers. The trailing edge of T1 is used to latch up the state of the RSC for address cycles, while the

trailing edge of T3 is used to latch up the RSC for data cycles. In addition, an ADDRESS CLOCK (+AC) and DATA CLOCK (+DC) are provided to enable the tri-state drivers attached to the RSC. The purpose of these clocks is to minimize the possibility of two devices on the bus simultaneously attempting to drive the bus to opposite polarities (because of skew problems). AC rises at the trailing edge of T0 and falls at the leading edge of T2 (nominally). DC rises at the trailing edge of T2 and falls at the leading edge of T0. The absence of DC on any data cycle indicates that ROMP will enter the stopped state on the next address cycle. While ROMP is in the stopped state, both AC and DC remain inactive. When this condition occurs, the bus contains invalid information, and no device should attempt to use the bus. The presence of DC on any cycle indicates that ROMP will not be in the stopped state on the next cycle. DC can be latched on the trailing edge of T3, and AC can be latched on the trailing edge of T1.

Figure 11 shows the timing for RSC clocks.

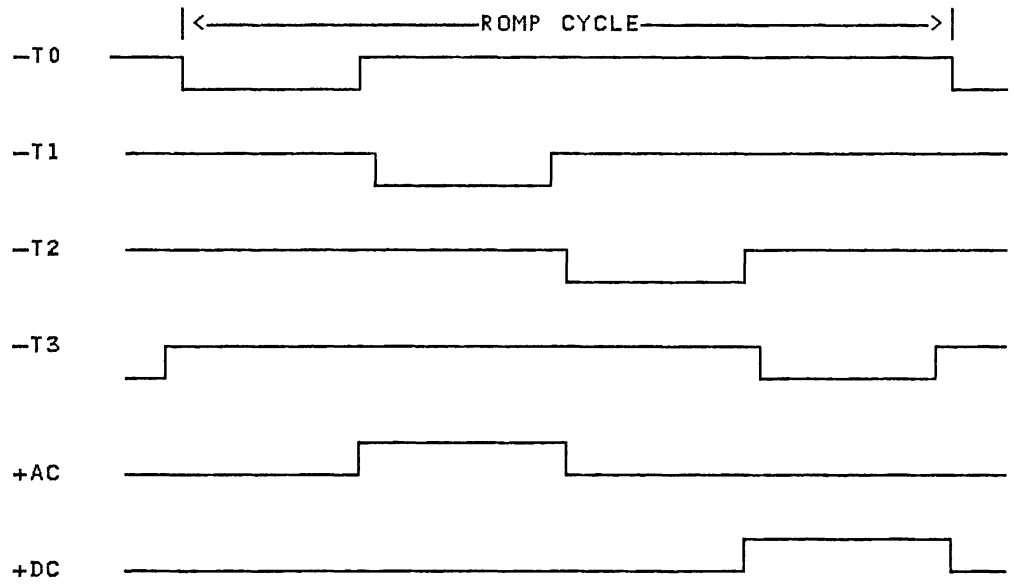


Figure 11. RSC Clock Timing

5.4 BUS OPERATION

5.4.1 Data Alignment

Replies to storage read requests are always 32-bits aligned on word (32-bit) boundaries. The low order two address bits are ignored. The maximum data transfer length on the RSC is 32 bits.

A storage write request must have its data portion properly aligned. For a 32-bit write, the most significant bit is placed on DAL00 and the least significant bit is placed on DAL31. For halfword and byte writes, the data to be written must be aligned as shown in the table below:

Transfer Type	Two Low-Order Bits	Data Position
Halfword	0X	DAL00-DAL15
	1X	DAL16-DAL31
Byte	00	DAL00-DAL07
	01	DAL08-DAL15
	10	DAL16-DAL23
	11	DAL24-DAL31

I/O addresses are considered to be device addresses, not byte addresses. The associated data transfers are 32-bits.

5.4.2 Bus Arbitration

The RSC arbitration mechanism consists of two linear-priority daisy-chains, the Address Grant chain and the Data Grant chain. Arbitration on the Address Grant chain takes place from the start of T0 to the end of T3. Arbitration on the Data Grant chain is from the start of T2 to the end of T1. Devices which issue requests participate in address arbitration, and devices which issue replies participate in data arbitration. Figure 12 on page 103 shows the timing for address and data cycle arbitration.

Each device which participates in address arbitration has a pair of pins, ADDRGRTI and ADDRGRTO which are connected in series with the other devices in that chain. An inactive ADDRGRTI forces an inactive ADDRGRTO. The highest priority device on the chain has its ADDRGRTI pin tied active. ROMP is by definition the lowest priority device on the chain, and does not have an ADDRGRTO. Each device also monitors the lowest order output of the Data Grant chain. A device which needs to transmit a request sets its ADDRGRTO inactive after the start of T0. If at the end of T3 it has an active ADDRGRTI, it assumes control of the RSC the following cycle. Successful arbitration for the address cycle also implies responsibility for the data cycle, unless the low-order DATAGRTO signal is inactive, which implies that some

device has requested use of the data cycle for a reply. If the data cycle is obtained for a read request, an IDLE packet should be issued. A write request may have its data portion preempted by a reply. This is necessary to avoid lockup conditions, and should cause a retry.

Devices which issue replies each have a DATAGRTI pin and a DATAGRTO pin. An inactive DATAGRTI will force an inactive DATAGRTO. These devices are chained, with ROMP supplying the input to the top of the chain. This allows ROMP to utilize the data cycle to issue a reset packet. A device which needs to send a reply forces its DATAGRTO inactive at the beginning of T2. If its DATAGRTI is active at the end of the next T1, then it assumes control of the RSC for the following data cycle, and transmits its reply. The low end of the chain is sent to all devices which perform address arbitration.

The low-order DATAGRTO may be used in another way. Because this line indicates whether a reply is being requested, if inactive it indicates that a WRITE DATA parcel is not being transmitted. This can be used by RSC receive logic to provide an early indication that a write data cycle has been preempted by a reply.

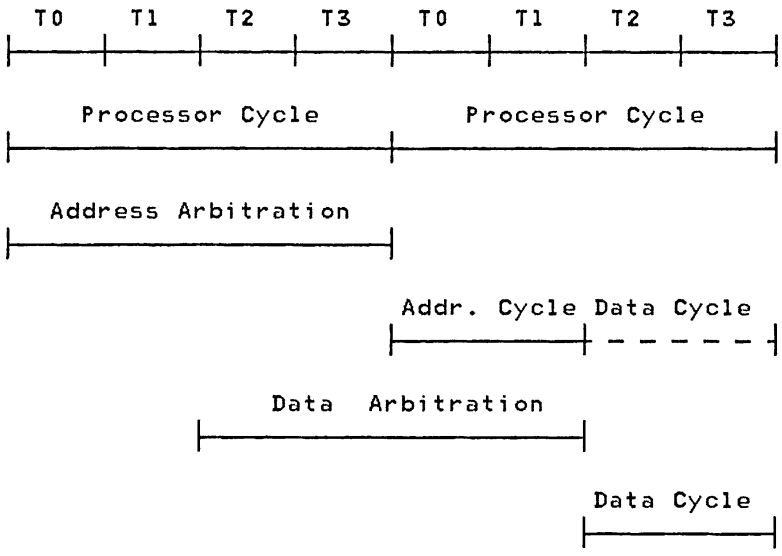


Figure 12. Bus Arbitration Timing

5.4.3 Read Request

A device which reads data from storage arbitrates for the bus as described in "Bus Arbitration" on page 102 using the ADDRGRT chain. At the leading edge of T0, the device places the address packet and its TAG ID on the bus and releases control of its outgoing ADDRGRT line. The device may also begin arbitration for the next address cycle if necessary. Storage latches the address and length information and the TAG on the trailing edge of T1. The device then releases the address/data bus and the tag bus. Storage examines the TAG to determine whether it is a valid request. If it is valid, storage proceeds to access the array and to arbitrate for a reply transfer using the DATAGRT chain. During T2 and T3 storage holds its ACKA signal active to indicate that it accepted the read request. If storage is busy and cannot accept the request it activates NAKA, and the device must retry the request. If storage detects a parity error on incoming information, it activates both ACKA and NAKA. The requesting device may retry the request or signal an error condition. At the leading edge of the next T0, storage releases the ACKA and NAKA lines, and the processor drives it to an inactive level. When the access is complete, and arbitration for a data bus cycle is successful, storage places its reply data on the address/data bus and the TAG ID on the TAG bus. The device latches the reply data and the TAG and takes appropriate action. Figure 13 shows the timing of a storage read request.

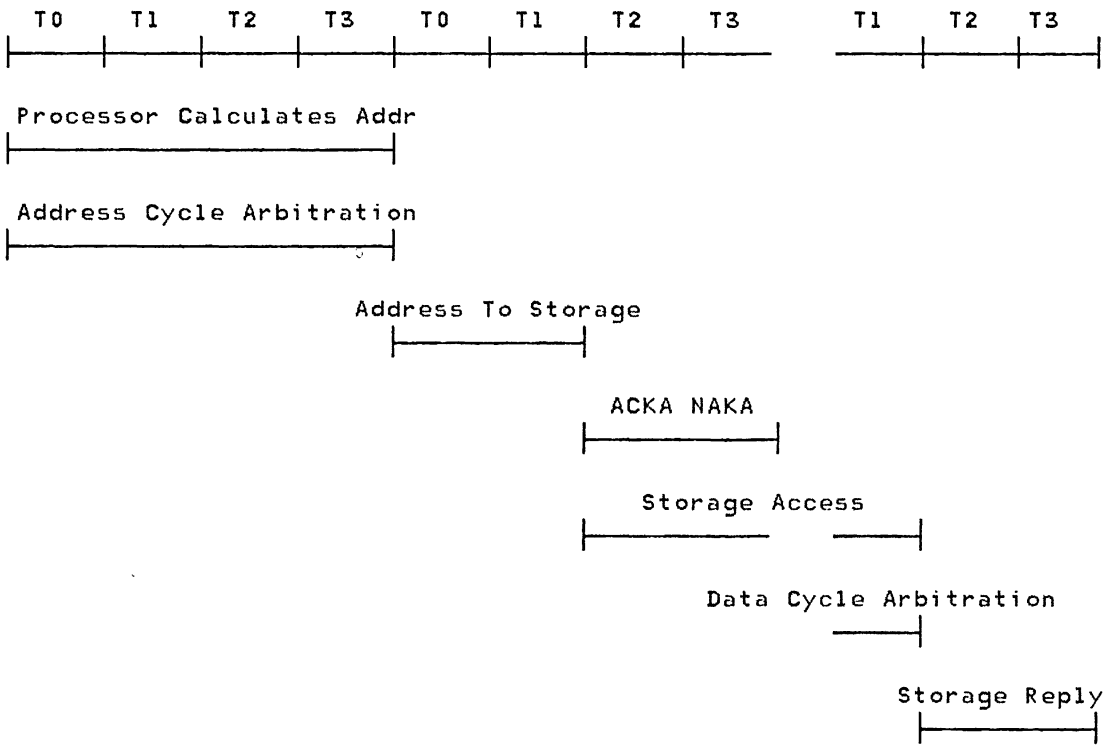


Figure 13. Read Request

5.4.4 Write Request

Writes to storage require two bus cycles, the first being an address cycle and the second being the very next data cycle. The device which is to do the write must arbitrate for both the address cycle and the data cycle by using the ADDRGRT and DATAGRT arbitration chains. If the device has successfully arbitrated for the address cycle and the data cycle is also available, the write operation can be placed on the bus. During the address cycle the device places the write address and length information on the data/address bus, and the write address tag on the tag bus. The ADDRGRT output is released at the beginning of the address cycle if the device does not need the next address cycle for another transfer. Storage will latch the address and tag information at the trailing edge of T1, and respond with an ACKA signal. If storage is busy, it will instead respond with a NAKA. In the case of a parity error storage will activate both ACKA and NAKA (see Figure 14). During the data cycle, the device which is doing the write places the data on the bus and the write tag on the tag bus. Storage latches the data on the trailing edge of T3 and checks parity on the data. If an error is detected, storage activates

ACKD and NAKD and the device may retry the transfer. Both the address and the data must be re-transmitted, even if only one error occurred.

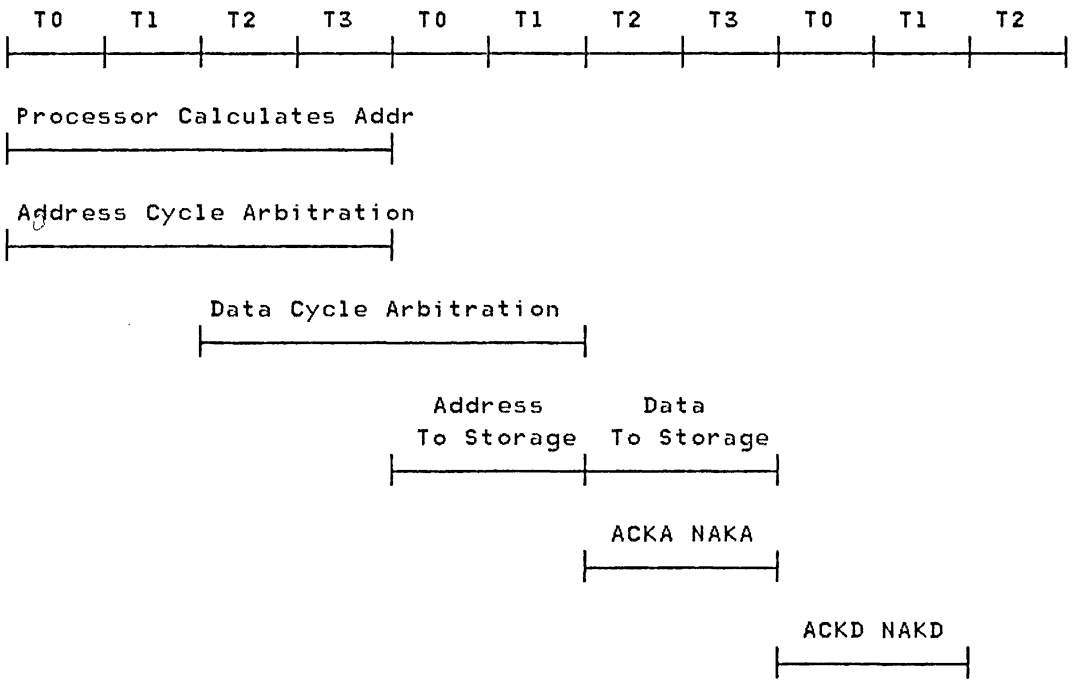


Figure 14. Write Request

5.4.5 Error Handling

When a device detects a parity error on the incoming data or address, it activates the ACKA and NAKA or ACKD and NAKD lines (whichever are appropriate) which causes the sending device to retry the transfer. A solid error will cause an endless retry condition so a timeout mechanism in the processor causes a machine check. See "Machine-Check Errors" on page 121 for more information.

If no device responds to a bus request generated by ROMP, the transfer will be tried two more times and if no response is detected, the transfer will be cancelled and bit 29 or bit 30 (whichever is appropriate) will be set in the PCS (See "Program-Check Errors" on page 124) and a program check will occur.

5.4.6 Idle Mode

Any device which gains control of the RSC but has no valid parcel to transmit must send an IDLE packet. An idle packet has a tag field of all zeros, and the address/data bus is unpredictable.

5.4.7 Reset

The RESET packet is a broadcast command sent by ROMP during a data cycle after a processor reset or after the processor detects a machine check condition (other than a machine check caused by the -TRAP input) and the check stop mask is a one. In the case of a machine check, the processor will reset certain registers and attempt to save the program status into main storage (beginning at address X'170'). In order to accomplish this , all outstanding storage references must be cleared. ROMP will gain control of the RSC during the data cycle by pulling DATAGRTO inactive and sending a RESET packet out on the TAG bus on the next data cycle. Whenever any device on the RSC detects a reset tag during the data cycle it must clear out any pending operations.

The following rules must be followed during the two cycles following a RESET packet to insure that the RSC remains properly defined:

1. Any device which was arbitrating for the address cycle following the RESET packet should place its request on the RSC during the address cycle. This request should be ignored by all devices. In addition, the transmitting device should expect no response to the request, and should reset the request thereafter.

2. Any data cycle arbitration taking place may complete. However, no device should transmit during the data cycle following a RESET packet, regardless of the results of the data cycle arbitration or any data cycle transmission implied by the address cycle arbitration. ROMP will always place an idle packet on the RSC during this data cycle.

3. Address cycle arbitration is cancelled after the RESET packet is received.

5.4.8 Illegal ACKD/NAKD Responses

Certain ACKD/NAKD responses are illegal, and are treated as error conditions by the device attempting the transfer. These illegal

responses to a data cycle transfer are defined below.

1. Responding busy to a reply. Devices on the RSC can not respond busy (ACKD inactive, NAKD active) to a reply. A busy response to a reply is treated as an error condition by the sending device. Error reporting and recovery by the sending device is design-dependent.

2. Responding busy to the data cycle of a store when the address cycle request of the store was accepted. Devices on the RSC can not respond busy to the data cycle of a store when the address cycle request was accepted. This response is treated as an error condition by the sending device. Error reporting and recovery by the sending device is design-dependent.

5.4.9 Engineering Note: ROMP Response To Illegal ACKD/NAKD Responses

Busy responses to replies do not apply to ROMP, since ROMP can not generate replies. If ROMP receives a busy response to the data cycle of a store, when the address cycle request of the store was accepted, the store operation will be retried a minimum of 128 times. If the device fails to accept the store, the store is terminated, and a machine check interrupt occurs. The RSC Timeout bit (bit 21) in the MCS is set to one.

5.4.10 Hold Time-Out Counter

The HOLD signal is used by devices on the RSC that can potentially interfere with ROMP access of the RSC. Devices on the RSC can prevent ROMP from accessing the RSC, thereby causing a machine check error due to the unavailability of the RSC. ROMP utilizes an internal time-out counter to detect the unavailability of the RSC. This counter is started when ROMP begins waiting for data or instructions from storage. The time-out counter is incremented by one during each cycle that ROMP is waiting, and is reset to zero when the reply is received. If this counter reaches a count of 128, a machine-check occurs.

If there are other devices on the RSC (DMA controllers, bus converters, etc.) they can activate the HOLD line to cause incrementing of the time-out counter to be inhibited while they are accessing storage on the RSC. The time out counter is not incremented while the HOLD line is active.

This signal is typically used by devices that are monopolizing the RSC for long periods of time, such as a DMA controller transferring many bytes of data in burst mode.

5.4.11 Storage Protection and Address Translation

ROMP provides a means of implementing storage protection and/or address translation. A detailed discussion of these subjects is contained in "Storage Controller Functions" on page 134. Whenever a device (either ROMP or any I/O device on the RSC) generates a storage reference, an address is sent out on the RSC. The address is contained in DAL08 through DAL31. Control information is contained in DAL00 through DAL06. If DAL04 is a one (active), storage protection is enabled. If DAL05 is a one (active), address translation is enabled. The actual storage protection and address translation hardware (if any) exists external to the processor. Assuming that protection or translation is installed, the storage controller does the necessary checking, and provides the appropriate response when a reply is generated. In the case of a read access, storage will generate a reply to the original requestor. When no exception condition exists, data is placed on the bus, the tag ID of the requestor is placed on the tag bus, and +EXCEPTION is driven to a zero (inactive). If an exception condition exists, the reply is generated in a similar manner, except +EXCEPTION is set to a one (active) and the data which is placed on the bus is not used but must have good parity.

Whenever a write access is attempted and storage protection or address translation is enabled, a reply must be generated to the original requestor, just as was done for a read access. The tag bus will contain the tag ID associated with the write address of the original requestor. The +EXCEPTION line will be driven inactive when no exception exists, and will be driven active when an exception does exist. In either case, the content of the data bus is unimportant, but must have good parity.

For systems which do not implement storage protection or virtual addressing, the +EXCEPTION line is not required. However, +EXCEPTION can still be used with replies to ROMP to report addressing exceptions. If ROMP receives a reply with +EXCEPTION active, and address translate and storage protect are disabled, then a program check occurs. PCS bits 25 (program check with unknown origin) and 30 (data address exception) are set. If the +EXCEPTION line is not used, the input to ROMP must be tied inactive.

When storage accesses are generated by ROMP, DAL06 will be active if the processor is in problem state and inactive if in supervisor state. This bit may be used to enhance the storage protection scheme by providing different types of access authority, such as read/write, read only, or execute only. See "Storage Controller Functions" on page 134 for additional information.

Only storage accesses can involve storage protection and/or address translation; i.e., all programmed I/O (PIO) accesses generate real addresses.

5.5 STORAGE CHANNEL I/O PIN SUMMARY

5.5.1 Storage Channel I/O Pin Summary for Processor

Signal Name	Function	Number of I/O Pins
+DAL00 Thru +DAL31	Addr/Data Bus	32
+DALP0	Odd Parity on DAL00-DAL07	1
+DALP1	Odd Parity on DAL08-DAL15	1
+DALP2	Odd Parity on DAL16-DAL23	1
+DALP3	Odd Parity on DAL24-DAL31	1
+TAG0 Thru +TAG4	Device ID	5
+TAGP	Odd Parity on TAG0-TAG4	1
-NAKA, -NAKD	Transfer Rejected	2
-ACKA, -ACKD	Transfer Acknowledged	2
+ADDRGRTI	Address Cycle Grant In	1
+DATAGRTI	Data Cycle Grant In	1
+DATAGRTO	Data Cycle Grant Out	1
-HOLD	Hold Time-Out Counter	1
+EXCEPTION	Address or Protection Exception	1
+AC	Address Cycle Clock	1
+DC	Data Cycle Clock	1
-T0, -T1 -T2, -T3	System Clocks	4
	Standard Total	57

+ADREXT0 Thru +ADREXT7	Address Extension Bus	8
+ADREXTP	Odd Parity on ADREXT0 thru ADREXT7	1
	Total With Address Extension	66

5.5.2 Storage Channel Pin Summary for a Typical RSC Component

Signal Name	Function	Number of I/O Pins
+ADDRGRTO	Address Cycle Grant Out	1
+SYSTEM POR		1
	Processor Signals Previously Defined	57
	Total	59

Storage units on the RSC do not require the Address Grant input or generate the corresponding output since storage does not transmit addresses.

5.6 ROMP STORAGE CHANNEL TIMING RELATIONSHIPS

The timing relationships in this section show a sample ROMP instruction sequence, assuming interleaved storage and two cycle storage access (or address translation with one cycle storage access). Figure 15 on page 112 defines the symbols and names used in Figure 16 on page 113 through Figure 19 on page 116.

As shown in Figure 16 on page 113, the first request (REQ1, having a tag of six) is placed on the RSC during the the address cycle of the second ROMP cycle. Storage accepts the request, as indicated by -ACKA being active, and -NAKA being inactive in the following data cycle. Two cycles later, on the data cycle of the fourth ROMP cycle (see Figure 17 on page 114) storage replies with REP1 which is accepted by ROMP (as indicated by -ACKD being active, and -NAKD being inactive in the following address cycle). Storage

¬¬¬¬¬	High
_____	Low
HHHHH	High Impedance
<===>	Bus is valid
REQn	nth RSC request
REPn	Reply to nth request
+EXCP	+EXCEPTION
+DAL	Data/Address Lines
+DATAGRTI	Data Grant in to ROMP

Figure 15. Signal Definitions

arbitration for a reply is indicated by the +DATAGRTI input to ROMP going inactive in the cycle preceding the reply.

Note that the replies are coupled to the requests as described in "ROMP Storage Channel" on page 92 (i.e. REQ1 has a tag of six and REP1 has a tag of six). Given that there are 5 tag lines, a maximum of 32 requests can be outstanding at any one time.

An interleaved storage configuration allows ROMP to transfer most requests across the channel on consecutive cycles. This allows the next three requests (REQ2, REQ3, REQ4) to be accepted by storage during ROMP cycles 7, 8, and 9 see (Figure 18 on page 115). Storage replies to the requests during the data cycle of ROMP cycles 9, 10, and 11. REQ5 (see Figure 19 on page 116) is attempted during the tenth ROMP cycle, but is not accepted because storage responded busy (as indicated by -ACKA being inactive, and -NAKA being active during the following data cycle). This request is retried on the next available address cycle and is accepted. Note that the +DATAGRTI input to ROMP is driven inactive by storage in the cycle preceding each storage reply.

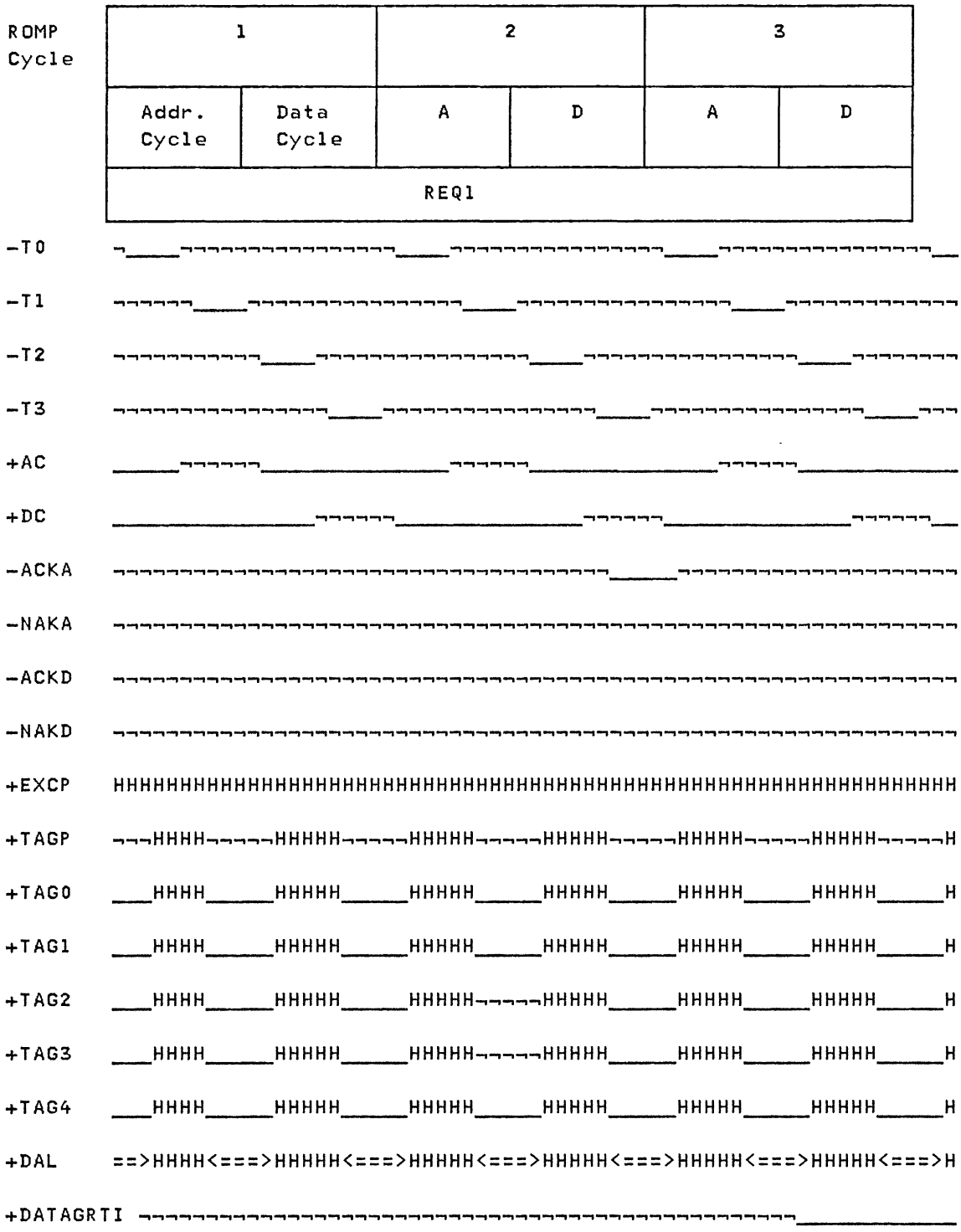


Figure 16. RSC Cycles One Through Three

ROMP Cycle	4		5		6	
	Addr. Cycle	Data Cycle	A	D	A	D
	REP1					

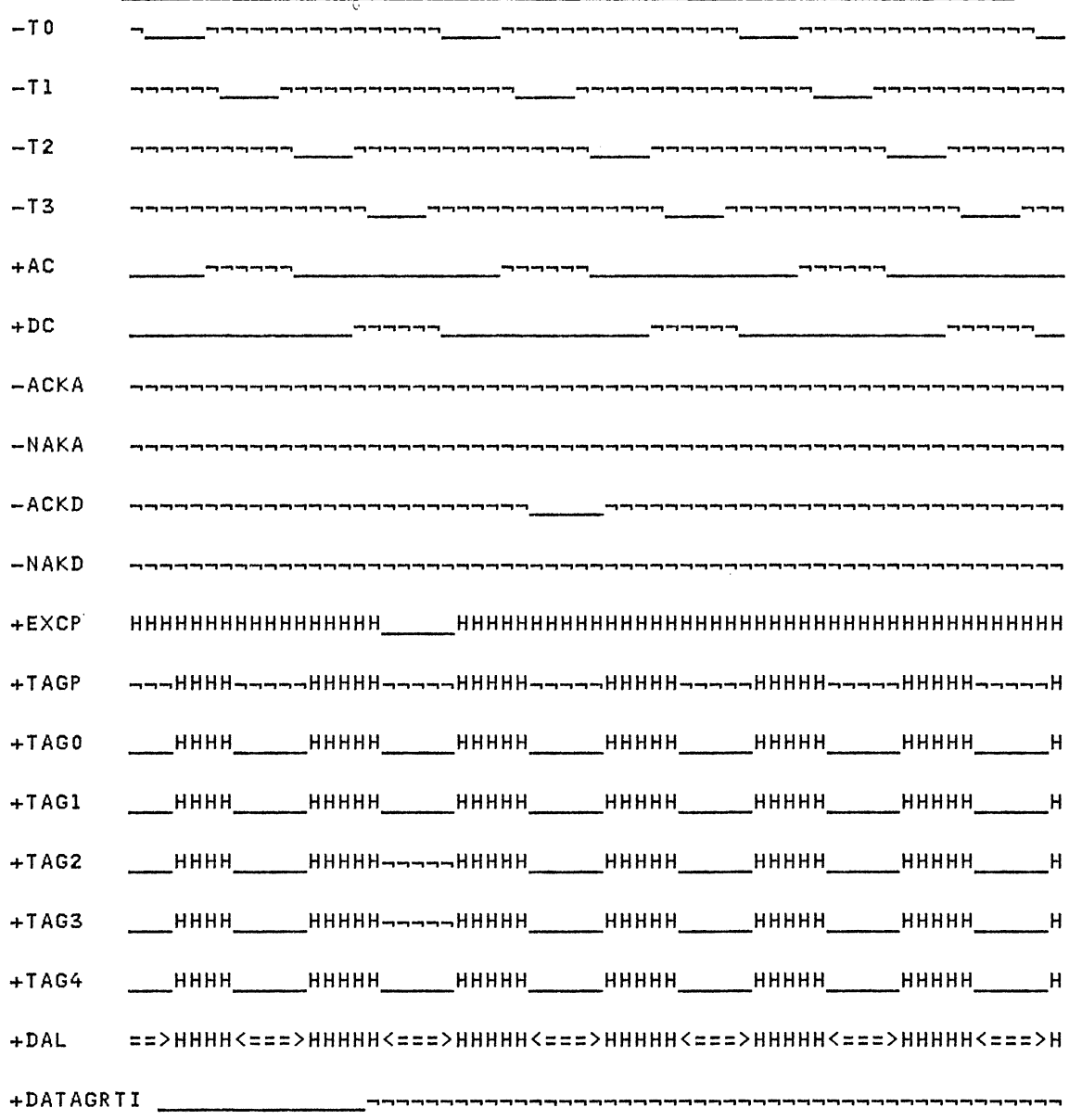


Figure 17. RSC Cycles Four Through Six

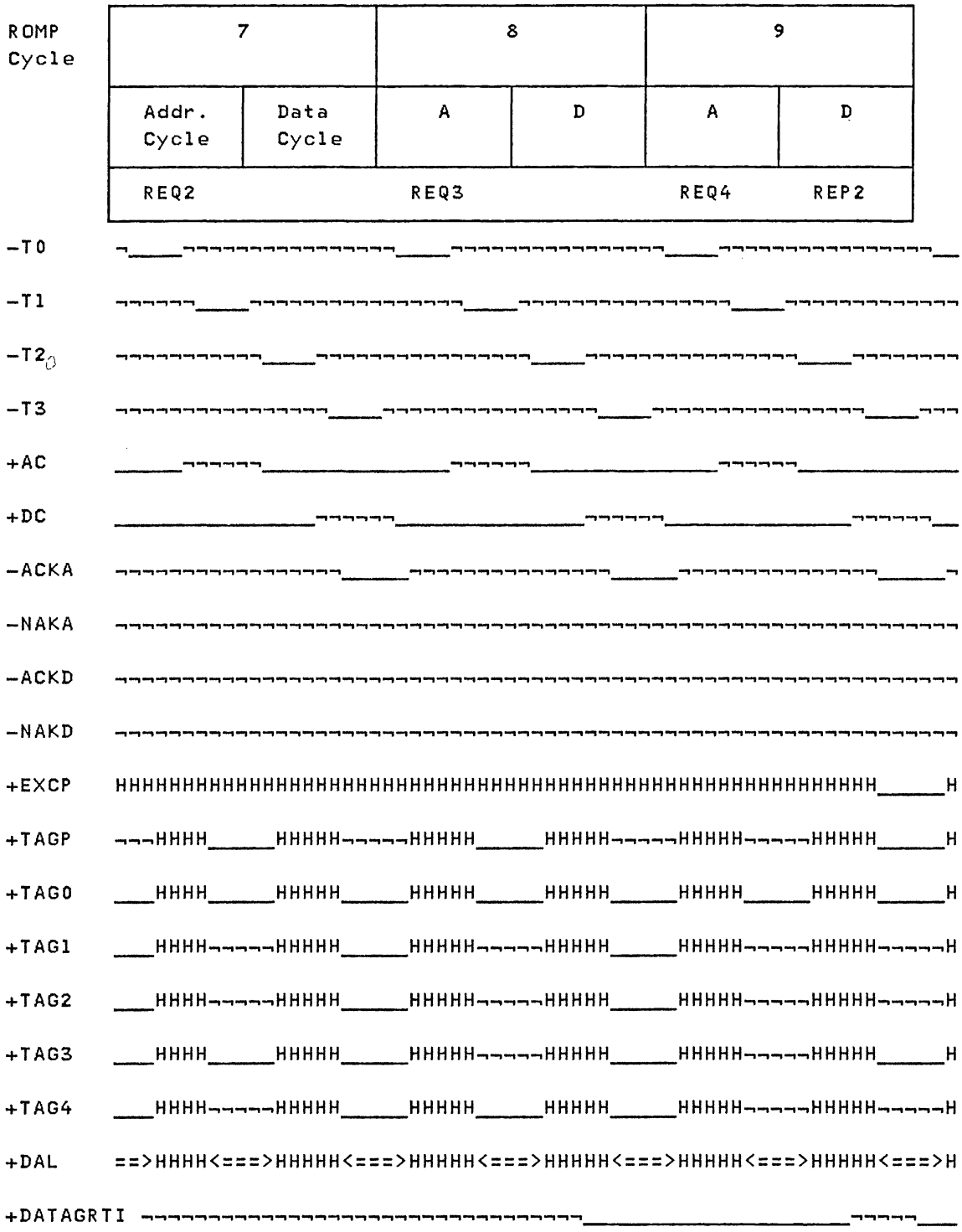


Figure 18. RSC Cycles Seven Through Nine

ROMP Cycle	10		11		12	
	Addr. Cycle	Data Cycle	A	D	A	D
	REQ5	REP3	REQ5	REP4	REQ6	

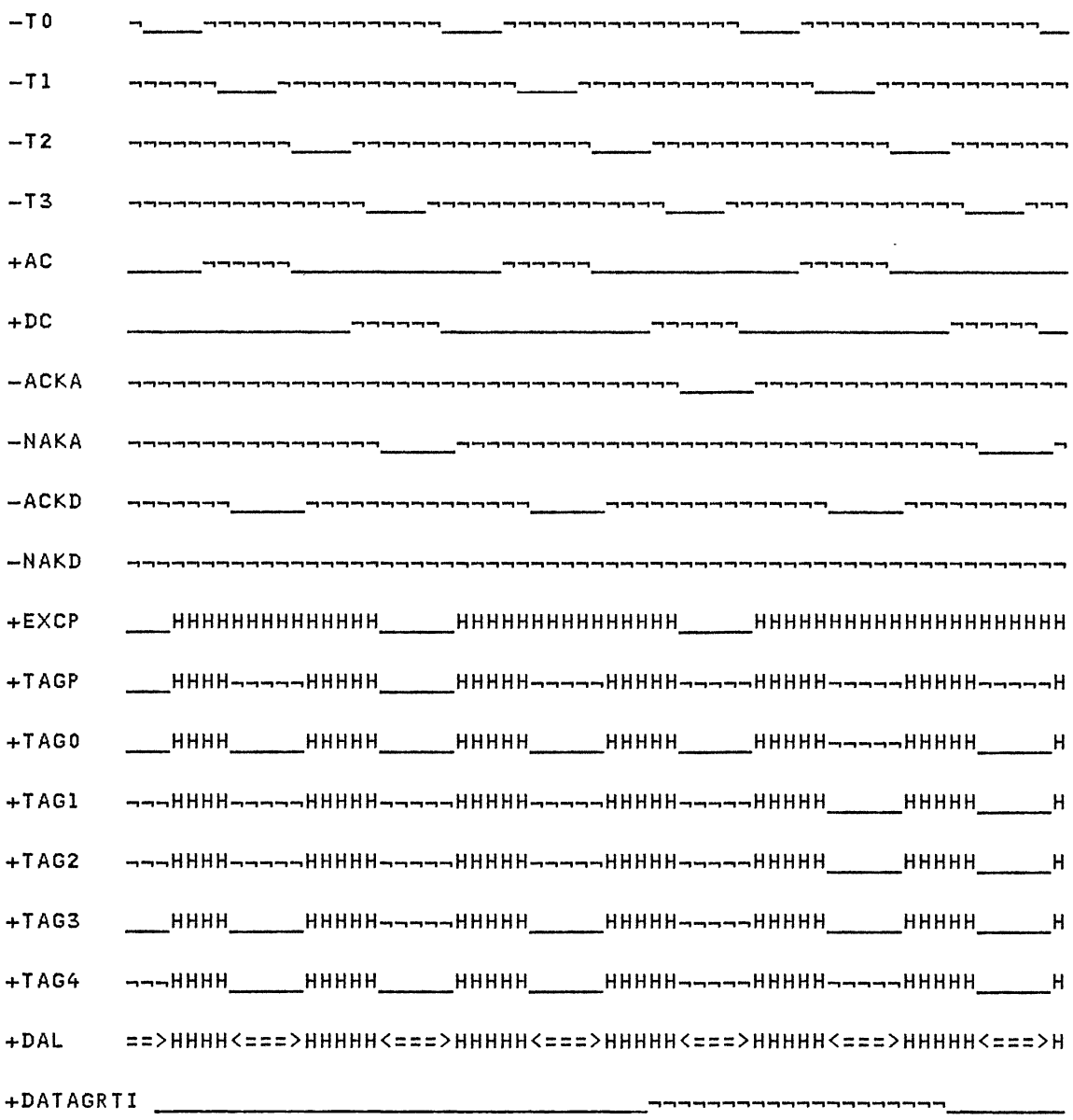


Figure 19. RSC Cycles Ten Through Twelve

6.0 INITIALIZATION

Initialization consists of a power-on reset (POR) sequence and initial program load (IPL). POR places the processor and system devices in a known state. IPL causes loading of program into main storage and program execution to begin.

6.1 POWER-ON RESET

When power is applied to the ROMP system, a POR signal is applied in order to bring the system to a defined state. POR consists of the following series of functions:

1. Processor and System Reset
2. Register Initialization
3. FAIL Pin State (active or inactive)

6.1.1 Processor and System Reset

Processor reset is accomplished by using the system POR signal to drive the +SCAN GATE input active. Since ROMP is initialized by resetting all internal latches, all scan inputs (-SCANI0 through -SCANI4) must be held inactive while +SCAN GATE is active. While +SCAN GATE is active, and all scan inputs are inactive, a sufficient number of system clock cycles must occur to reset all internal latches. The specific number of clock cycles required is defined in the ROMP E-Spec (see "ROMP Engineering Specification" on page 188).

After a sufficient number of clock cycles have occurred, +SCAN GATE can be driven inactive, and ROMP will be in a reset state. While +SCAN GATE is active, all scan-inputs to ROMP must be held inactive. Failure to hold all scan-inputs inactive will cause unpredictable results.

Processor reset consists of the following operations:

1. The execution of any current processing state is terminated.
2. Any I/O interrupt requests are cleared.
3. Any machine check and program check conditions are cleared.

4. If the processor is in the check stop state, the check stop condition is cleared.

Any system devices attached to the processor are initialized as required. This can be performed by connecting the individual device reset lines to the system POR signal.

6.1.2 Register Initialization And Diagnostics

The register initialization function places the processor registers in a defined state. The register initialization routine includes diagnostics which check a major portion of the control and data paths needed for instruction execution. No ROMP Storage Channel (RSC) functions are checked. No other system components are tested by the internal diagnostics. Initialization consists of the following operations:

1. The contents of all general purpose registers are set to zero. If any bit of any GPR is stuck-at-zero or stuck-at-one, register initialization does not complete successfully.

2. All defined system control register (SCR) bits are set to zero except the following:

   a. The Processor Priority, bits 29-31 of SCR 14 is set to 7.

   b. The Interrupt Mask, bit 23 of SCR 14, is set to one (disabled).

6.1.3 Fail Pin State

POR initializes the I/O pin -FAIL to an active state. If an error is detected during the register initialization microcode routine, the processor enters the check stop state (see "Executing, Wait, Check Stop, and Stopped State" on page 7) and the -FAIL pin remains active. If no errors are detected during register initialization, the I/O pin -FAIL is brought inactive at the completion of register initialization. This pin can be sensed by an external device to detect processor failure.

6.2 PROGRAM INITIALIZATION

Once POR is completed, the processor is in a state waiting to begin instruction execution. This state is indicated by the I/O pin (-IPL READY) going active. The processor will not load the IAR or begin instruction execution until it receives a signal (-IPL COMPLETE) indicating that storage has been loaded or that storage loading is not required for the system configuration.

6.2.1 Initial Program Load

This step is optional and required only when the system contains no ROS and initial programs are loaded into RAM from an I/O device on the RSC.

When the POR sequence is completed, the I/O pin -IPL READY is brought active by the processor to indicate that the processor is ready either for storage to be loaded or to begin program execution. If an I/O device is used to load programs, it senses this line and transfers instructions to main storage via DMA operations. Once the device has completed loading main storage, the processor I/O pin -IPL COMPLETE is brought active by the IPL device to cause an IAR load to occur. -IPL READY is brought inactive by the processor one cycle after -IPL COMPLETE goes active. -IPL COMPLETE is then ignored by the processor until the next POR sequence occurs.

6.2.2 IAR Load

The initial IAR load occurs when the -IPL COMPLETE line is brought active after POR. When this pin is brought active, the IAR is from location 00000000 in main storage. Program execution then begins from the address loaded into the IAR.

6.2.3 Engineering Notes: Initialization

1. Power-on reset leaves the contents of main storage in an unpredictable state except for the locations which are initialized by the initial program load function.

2. If ROS is used to contain initial programs, the processor signal -IPL COMPLETE can be connected to -IPL READY, or tied active to cause the IAR load to occur automatically when POR is completed.

3. It is not required that storage be implemented at location 00000000, but storage must respond to this address, and it must reply with an initial IAR for the processor. This may be done by mapping the address 00000000 to a predefined storage location, or by providing a hardware register which responds to this address.

7.0 RELIABILITY AVAILABILITY AND SERVICEABILITY

7.1 RAS FACILITIES

RAS Facilities provide for:

1. Detection of processor errors.
2. Detection and isolation of program-related errors.
3. Decreased exposure to data loss and error situations.

7.2 SYSTEM ERROR DETECTION AND REPORTING

7.2.1 Internal Diagnostics

The processor executes an internal microcode routine to perform register initialization when a processor reset occurs. Successful completion of the register initialization routine provides reasonable confidence that the processor is functional for instruction execution. The internal microcode diagnostic does not verify any RSC functions, or other system components.

The I/O pin -FAIL is initialized to an active state during POR. If no errors are detected during register initialization , -FAIL is brought inactive. If an error is detected, the processor enters the Check Stop state and -FAIL remains active. This ensures that a failure condition will be indicated if the processor is unable to execute the register initialization microcode.

7.2.2 Machine-Check Errors

Machine-check errors are those errors which are most probably caused by hardware malfunctions.

7.2.2.1 Machine-Check Error Handling

Upon the detection of a machine-check error condition, other than an I/O trap, all current processor activity is halted, regardless of that activity. If the detected error is an I/O trap, the processor will complete its current activity before servicing the error. I/O traps are reported by activating the -TRAP input. The processor then takes one of two courses of action, depending on the value contained in the Check Stop Mask.

If the Check Stop Mask has a value of zero, the processor enters the Check Stop state when any machine check error is detected (including an error reported by -TRAP). This preserves the state of internal processor latches for inspection by a support processor. Refer to "Support Processor Facilities" on page 142 for a description of support processor functions. The I/O pin -FAIL is brought active to indicate a failure.

If the Check Stop Mask has a value of one, and a machine check error is detected, other than one caused by the -TRAP interrupt input, a reset packet will be sent on the RSC to clear any pending RSC operations. If the machine check interrupt is caused by the -TRAP interrupt input, no reset packet will be sent on the RSC. If the Check Stop Mask has a value of one, the processor saves the current program status in the old program status location in the Machine Check Old/New PS pair (beginning at location X'170'). The program status for servicing the error is then loaded from the new program status location, with the exception of the Condition Status, and the processor attempts to continue execution.

The machine check routine must execute a Load Program Status (LPS) instruction to return from the machine check error.

7.2.2.2 Machine-Check Status

The Machine-Check Status (MCS) provides a means for reporting hardware malfunctions. Information is provided to assist an error servicing routine in determining the type and source of the error.

The MCS is an eight-bit field in system control register 11. Upon the detection of a machine check error, appropriate bits of the MCS are set to ones (except for the Parity Check bit (bit 18) which is set whenever the processor receives a reply with invalid data parity).

The MCS is defined as follows:

Bit 16 RSC Check. Set to one when a device on the RSC detects invalid parity on a processor-generated transfer over an abnormally large number of retries. This bit is also set when the processor generates an interrupt to report an RSC retry which successfully corrected a parity error (See "Interrupt Control Status" on page 21).

Bit 17 Reserved.

Bit 18 Parity Check. Set to one whenever the processor receives a reply on the RSC with invalid data parity. This bit is set whether or not a machine check occurs.

Bit 19 Instruction Timeout. Set to one when the processor fails to receive an expected reply to an instruction fetch.

Bit 20 Data Timeout. Set to one when the processor fails to receive an expected reply to a data fetch.

Bit 21 RSC Timeout. Set to one whenever the processor has been unable to transfer a request on the RSC over an abnormally large number of cycles, and no parity errors have been signalled. The request may be unsuccessful due to busy responses or unsuccessful arbitration.

Bit 22 I/O Trap. Set to one when an I/O device signals a trap condition.

Bit 23 Reserved.

The MCS is cleared when a Load Program Status (LPS) instruction is executed to return from the Machine Check level.

The MCS provides a summary of processor conditions which are present when a machine check error is detected. Thus, it is possible that multiple bits of the MCS are set upon detection of an error. For example, if bits 16,19, and 20 of the MCS are set, the processor failed to receive a reply to both an instruction and data fetch. In addition, a device on the RSC detected invalid parity on a processor-generated request. In this case, invalid parity on the request prevented the processor from successfully transferring both instruction and data requests.

7.2.3 Engineering Note: RSC Retry

Transfers from the processor to other system components on the RSC are automatically retried by the processor, if the first transfer

attempt fails. The number of retries are dependent on the response to the first transfer attempt, and the processor implementation. This implementation retries failing transfers based on the response from the first transfer attempt as defined below:

1. No device on the RSC responds (no ACKA or NAKA) to a processor generated instruction or data fetch. The processor retries the transfer twice. If no device responds to the two retries, the transfer attempt is terminated, and a program check interrupt occurs. The Instruction Address Exception bit (bit 29) or the Data Address Exception bit (bit 30) in the PCS is set to one (based on whether the failing request was an instruction or data fetch) to indicate the type of transfer.

2. A device on the RSC responds busy (NAKA, but no ACKA) to a processor generated instruction or data fetch. The processor retries the transfer a minimum of 128 times. If the device fails to accept the transfer, the transfer attempt is terminated, and a machine check interrupt occurs. The RSC Timeout bit (bit 21) in the MCS is set to one.

3. A device on the RSC responds with a parity error indication (ACKA and NAKA) to a processor generated instruction or data fetch. The processor retries the transfer a minimum of 128 times. If the device fails to accept the transfer, the transfer attempt is terminated, and a machine check interrupt occurs. The RSC Check bit (bit 16) in the MCS is set to one.

7.2.4 Program-Check Errors

Program-check errors are those errors which are most probably caused by software errors.

7.2.4.1 Program-Check Error Handling

Upon the detection of a program check error condition, the processor completes its current activity (instruction, timer, etc.), unless that activity caused the program-check condition. The processor then saves the current program status in the old program status location in the Program Check Old/New PS pair (beginning at location X'180'). The program status for servicing the error is loaded from the new program status location, with the exception of the Condition Status.

The program check routine must execute a Load Program Status (LPS) instruction to return from the program check error.

7.2.4.2 Program-Check Status

The Program-Check Status (PCS) provides a means for reporting certain programming errors. Reported program check errors include attempted execution of an unassigned or unimplemented operation code, the attempted execution of a privileged instruction with the Problem State bit (bit 21 of SCR 14) being a one, an improper data condition which is detected by the execution of a trap instruction, and attempted access of an invalid storage location.

The PCS is an eight-bit field in system control register 11. Upon the detection of a program check error, all bits of the PCS are set to zeros. The appropriate bits of the PCS are then set to ones.

The PCS is defined as follows:

Bit 24 Program check with known origin. Set to one when a program check occurs and the location of the causing instruction is determinable from the IAR in the old program status.

Bit 25 Program check with unknown origin. Set to one when a program check occurs and the location of the causing instruction is not determinable from the IAR in the old program status.

Bit 26 Program Trap. Set to one when a trap exception condition is generated by a trap instruction.

Bit 27 Privileged Instruction Exception. Set to one when the processor attempts to execute a privileged instruction and the Problem State Bit (bit 21 of SCR 14) is a one.

Bit 28 Illegal Operation Code. Set to one when the attempted execution of an unassigned or unimplemented operation code is detected.

Bit 29 Instruction Address Exception. Set to one when no device on the RSC responds to a processor instruction fetch request, or when a reply to an instruction fetch is accompanied by an invalid address indication.

Bit 30 Data Address Exception. Set to one when no device on the RSC responds to a processor data request, or when a device on the RSC responds to a data request with an invalid address indication. This bit is also set when an access to a privileged I/O device is attempted from problem state.

Bit 31 Reserved.

The PCS is cleared when a Load Program Status (LPS) instruction is executed to return from the Program Check level.

The detection of a program trap condition set both bits 24 and 26 to ones. The IAR in the old program status contains the address of the trap instruction.

The detection of a privileged instruction exception sets bits 24 and 27 to ones. The IAR in the old program status contains the address of the privileged instructions. If the privileged instruction is the subject of a branch with execute, the IAR in the old program status contains the address of the branch with execute instruction.

The detection of an illegal operation code sets both bits 24 and 28 to ones. The IAR in the old program status contains the address of the illegal operation. If the illegal operation is the subject of a branch with execute, the IAR in the old program status contains the address of the branch with execute instruction.

If a data address exception occurs, bit 30 of the PCS is set to one, and either bit 24 or 25 is set to one to indicate the meaning of the IAR in the old program status. If bit 24 is set to one, the IAR in the old program status contains the address of the instruction which attempted to access the invalid storage location. If the subject instruction of a branch with execute attempts to access an invalid storage location, the IAR in the old program status contains the address of the branch with execute instruction. If bit 25 of the PCS is set to one, the IAR in the old program status contains the address of the instruction which was executing when the data address exception was detected. If this instruction required the data which was accessed at the invalid address, it did not complete successfully.

Figure 20 on page 127 provides a summary of program check errors when address translation and storage protect are disabled. Figure 21 on page 128 provides a summary of program check errors when address translation or storage protect is enabled.

7.2.4.3 Programming Note: Instruction Restart

If address translation or storage protect is enabled and a data address exception occurs, the IAR in the old program status word always points to an instruction which can be restarted once the exception conditions are resolved. In most cases, attempted execution of the instruction causing the data address exception has no effect on the values contained in the general purpose registers (GPRs), system control registers (SCRs), or data in

storage. However, in the case of Load Multiple and Store Multiple, it is possible for several of the loads or stores to occur before the exception is detected. The processor does not restore all GPRs or storage to the state which existed prior to attempted execution of the instruction causing the data address exception. However, if a Load Multiple causes an exception, the Load Multiple base address register (RC) will be restored to its original value so that the Load Multiple instruction can be restarted.

Program Check Error	PCS Bits							
	24	25	26	27	28	29	30	31
1. Invalid instruction address.	1	0	0	0	0	1	0	0
2. Invalid data address.	0	1	0	0	0	0	1	0
3. Successful trap instruction.	1	0	1	0	0	0	0	0
4. Privileged instruction exception.	1	0	0	1	0	0	0	0
5. Illegal op-code.	1	0	0	0	1	0	0	0

Figure 20. Program Check Errors With Storage Protect And Address Translation Disabled

Program Check Error	PCS Bits 24	25	26	27	28	29	30	31
1. Invalid instruction address.	1	0	0	0	0	1	0	0
2. Invalid data address.	1	0	0	0	0	0	1	0
3. Successful trap instruction.	1	0	1	0	0	0	0	0
4. Privileged instruction exception.	1	0	0	1	0	0	0	0
5. Illegal op-code.	1	0	0	0	1	0	0	0

Figure 21. Program Check Errors With Storage Protect Or Address Translation Enabled

7.2.5 Simultaneous Program Check and Machine Check Errors

Certain hardware error conditions can result in both a program check and a machine check error, as the result of a single request. For example, a storage controller may generate both an exception reply and activate the -TRAP input to report an uncorrectable storage error (uncorrectable ECC error or parity error). The exception reply causes a program check, and the -TRAP causes a machine check. The exception reply is necessary to prevent the processor from using bad data from the request, and -TRAP is required to report hardware errors at the machine check interrupt level. If a machine check and program check error occur simultaneously, the processor will perform both a program check PSW swap, and then a machine check PSW swap. In order to guarantee that both the program check and machine check interrupts are properly handled, system devices using both an exception reply and -TRAP must report both errors simultaneously. Devices should activate -TRAP when the error is detected, and send an exception indication with the reply.

Once the program check and machine check PSW swaps are completed by the processor, the machine check interrupt handler will be executed. The machine check and program check interrupt handlers must be properly designed to handle this multiple error condition. The machine check interrupt handler can determine the reason for the interrupt by examining status bits in the MCS and status

register(s) in each system component. Once the source of the error has been isolated and logged, the machine check interrupt handler can complete, and should execute an LPS instruction to return from the machine check interrupt. Executing an LPS on the machine check level will clear all bits in the MCS, and return to the next highest priority level. In this case, a program check interrupt is pending so the machine check interrupt handler will return to the program check interrupt handler.

System software must be constructed so that there is sufficient information available to the program check interrupt handler to determine that the program check interrupt was due to the machine check error that was previously handled. If the machine check interrupt handler performed all of the steps necessary to service the error, then the program check interrupt handler should simply execute an LPS instruction to return from the program check level. Executing the LPS on the program check level will clear all bits in the PCS, and return to the routine that was executing when the error was detected.

Note that in this case, a single error causes both a machine check and a program check interrupt, and that both interrupts must be processed. The machine check interrupt handler will always be executed first, and will return to the program check interrupt handler by executing an LPS instruction. Once the program check interrupt handler completes, it returns to the routine where the error was detected by executing an LPS. Executing an LPS from the machine check level terminates processing on the machine check level, clears all bits in the MCS, and returns to the next highest priority interrupt. Executing an LPS from the program check level terminates processing on the program check level, clears all bits in the PCS, and returns to the next highest priority interrupt.

7.3 MULTIPLE OCCURRENCE OF ERRORS

The processor will enter the Check-Stop state, regardless of the value in the Check Stop Mask, under the following conditions:

1. The processor is servicing a machine-check error, and another machine-check error is detected.

2. The processor is servicing a machine-check error, and a program-check error is detected.

3. The processor is servicing a program-check error, and another program-check error is detected.

If the processor is servicing a program-check error and a machine-check error is detected, the machine-check error is handled as outlined in "Machine-Check Error Handling" on page 122.

8.0 MULTIPROCESSOR SYSTEM

8.1 GENERAL DESCRIPTION

The ROMP processor and RSC contain features which support a multi-processor configuration. A TEST & SET instruction allows control of shareable resources and the RSC, operating with a packet-switch protocol, allows commands and data to be directed throughout the system.

8.2 TEST & SET INSTRUCTION OPERATION

The Test & Set instruction, from the processors standpoint is a LOAD instruction with a special function code. This code instructs the storage unit to immediately follow the read operation with a byte write of all ones. The contents of the specified location are loaded into the specified register and the high-order byte in storage is set to all ones with the low-order byte unaltered. If a processor or task chooses to identify itself, it can then write a descriptive code in the low-byte, knowing it has control of that halfword as well as the resource under contention. When completed with the resource, the processor can then clear this halfword to again make the resource available.

8.3 MULTIPROCESSOR SYSTEM INTERCONNECTION

If two processors are connected to the same storage channel, performance will be less than that achievable with two single processors due to storage interference. Each processor should be able to use a large share of the available storage bandwidth and each would spend considerable time waiting if both were connected to the same storage channel. To more effectively use each processor, each should have its own private storage for most instructions and data. Each private storage contains instructions, data and separate PSW areas for interrupt and exception conditions. However, to effectively coordinate activities, both processors must be able to exchange commands and data. This can be done by a multi-port memory that can be accessed by each processor, a bus coupler, or a communications channel.

For the multi-port memory, a portion of the address space is shared by more than one processor. This is shown in Figure 22 on page 132.

A bus coupler can be used to isolate one processor memory system from another as shown in Figure 23 on page 132. Each device on the storage channel is assigned a tag or source address. The processor may generate a READ command for an address, known by the coupler to exist on another channel The coupler accepts the command, saves the tag, and retransmits the command on the other bus with a coupler tag. The storage unit will send data to the coupler, which the coupler will then send to the processor with the original tag.

In isolated systems, a communications technique may prove to be the best solution when multiple processors are required (See Figure 24 on page 133). This method uses formatted and addressed packets. Processor A formats and addresses a block of commands or data in its memory. It then instructs the communications adapter to transmit the block of data to another system. The receiving adapter need only recognize the destination code and store the data in the predefined input buffer for us by processor b. Both processors are then interrupted; one being notified that data has been received, and one that data has been sent.

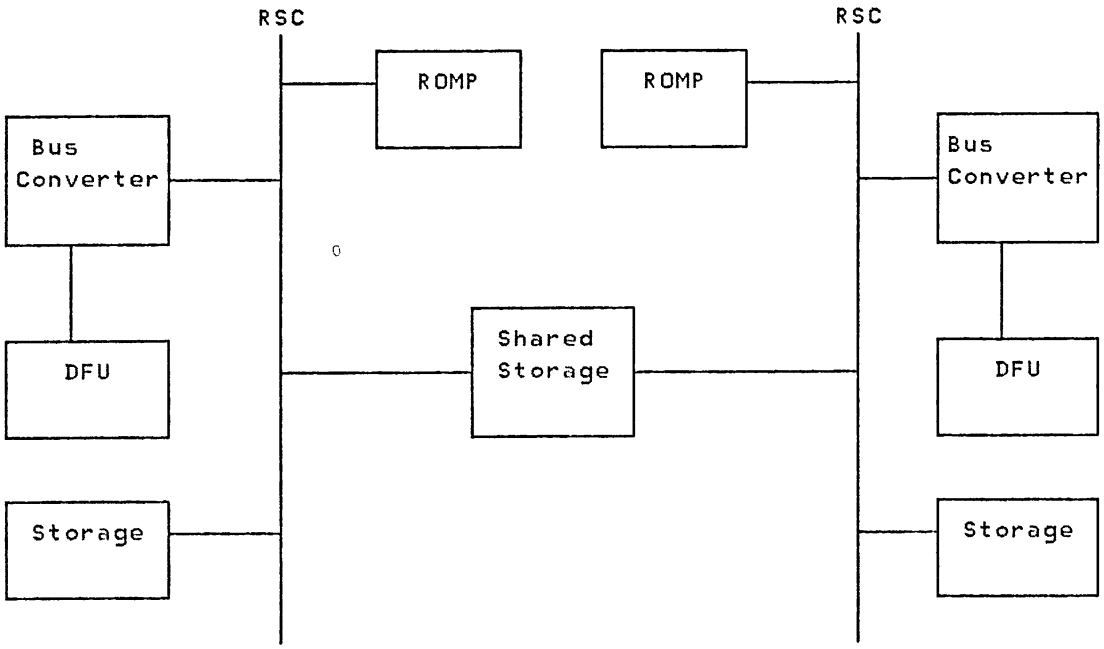


Figure 22. Multi-Processor Connection Via Common Storage

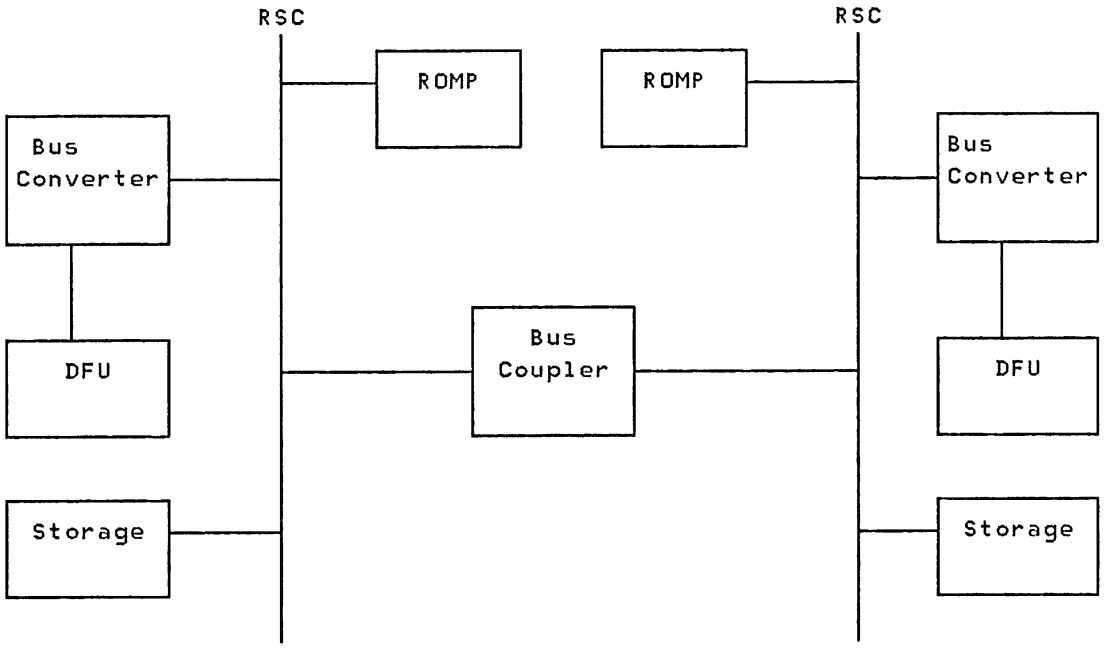


Figure 23. Multi-Processor Connection Via Bus Coupler

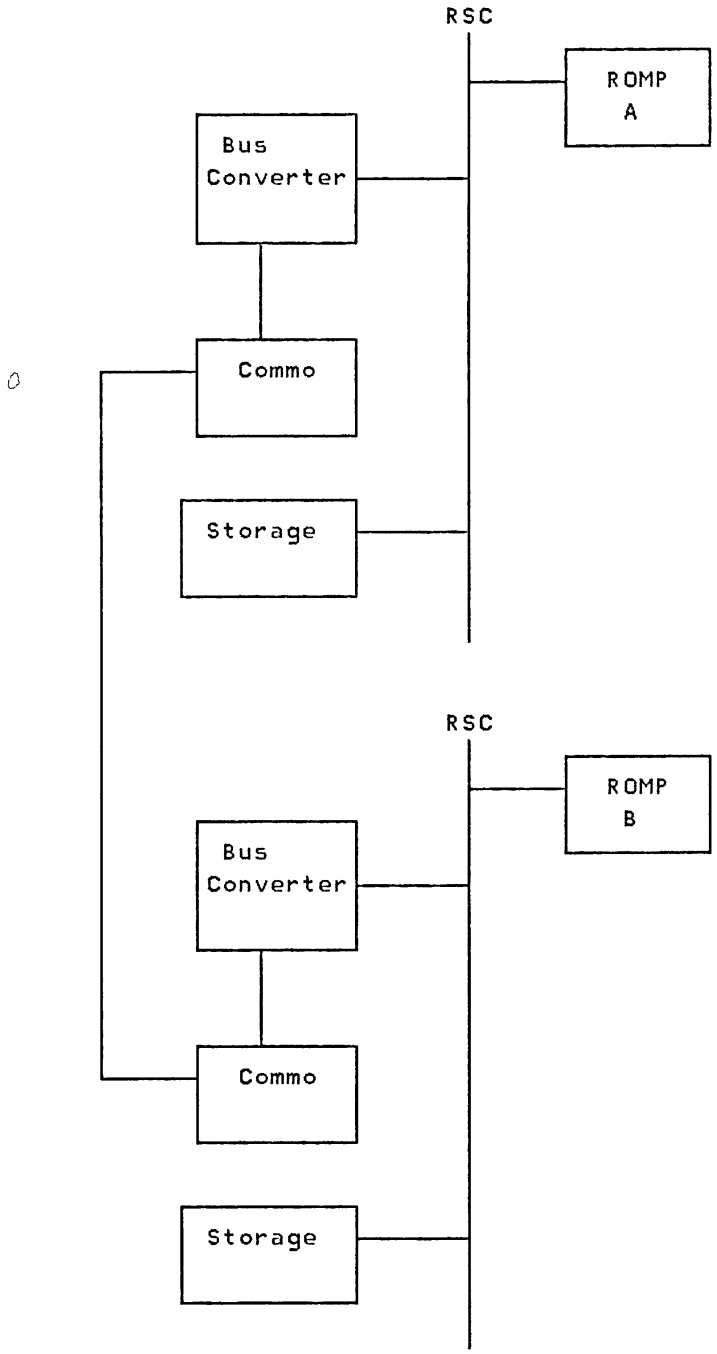


Figure 24. Multi-Processor Connection Via Communications Link

9.0 STORAGE CONTROLLER FUNCTIONS

This section is intended to illustrate the range of storage controller options which can be implemented in a ROMP system.

In any ROMP system, the processor, and possibly other devices will be requesting storage operations. These requests will be examined by one or more storage controllers and either accepted or not accepted (this is indicated by the ACK,NAK handshaking lines one RSC cycle after the request). A request will be accepted if the storage controller is available, and the address associated with the request is a valid address. Instruction fetches and loads require that the storage controller generate a reply at some later time. The reply contains the requested data and the tag which was sent with the request, indicating the destination of the reply. The reply can occur on any cycle after the request was received. The RSC can be utilized by other devices while a request is being processed by the storage controller. ROMP takes advantage of this by attempting to overlap storage accesses with instruction execution.

A simple ROMP system might contain a single storage controller and a single storage array. When a storage request arrives from the RSC, the storage controller would perform the required access, and if necessary, generate a reply on the RSC. Requests arriving before the storage array was free would receive a BUSY response. Figure 25 and Figure 26 shows the approximate timing for this type of controller. Figure 25 shows the timing for fast storage, and Figure 26shows the timing for slower storage. Note that Figure 25 shows the best possible performance, where the storage access time is equal to the processor cycle time.

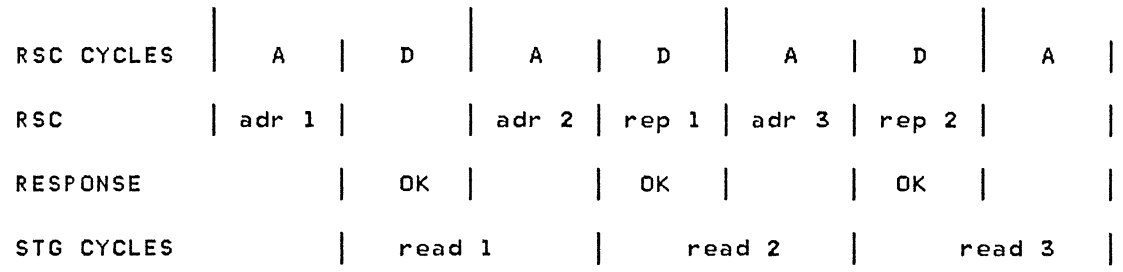


Figure 25. Storage Controller Timing With Fast Storage

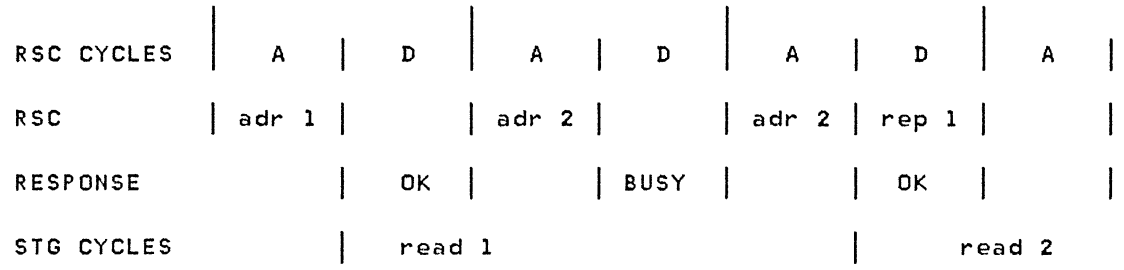


Figure 26. Storage Controller Timing With Slow Storage

The storage controller can also control multiple arrays. One reason for this might be to mix storage technologies (for example, ROS and RAM), or to control two interleaved arrays. An interleaved storage system utilizes one array for all even word addresses, and a second independent array for all odd word addresses. An interleaved storage system can provide an increase in the available storage bandwidth since the storage controller can overlap accesses of interleaved arrays. This will result in improved processor performance since a large percentage of processor generated storage references are for sequential word addresses.

Multiple storage controllers can also be used, where each controller has one or more storage arrays. This type of system allows each controller and storage array to operate independently of the other storage controllers. The design of the individual controllers is also simplified if each controls only one array.

It is important to note that the RSC design allows additional pipelined stages to be placed in series with the storage access stage, that do not reduce the bandwidth for instruction prefetches. However, these stages do increase the latency for non-overlapped accesses such as the first instruction fetch following a successful branch, and all data accesses. The ability to add such a stage can be beneficial when designing a storage controller employing error correcting codes (ECC). ECC delays are usually added in series with the array access time, and frequently increases storage access time. In a ROMP system, a possible alternative is to provide a separate pipelined stage to perform ECC checking. Depending on the amount of time required to perform the ECC check, it might be advantageous to overlap the ECC check with the next storage cycle, and delay the reply by a full RSC cycle. Figure 27 shows the timing for this type of ECC checking.

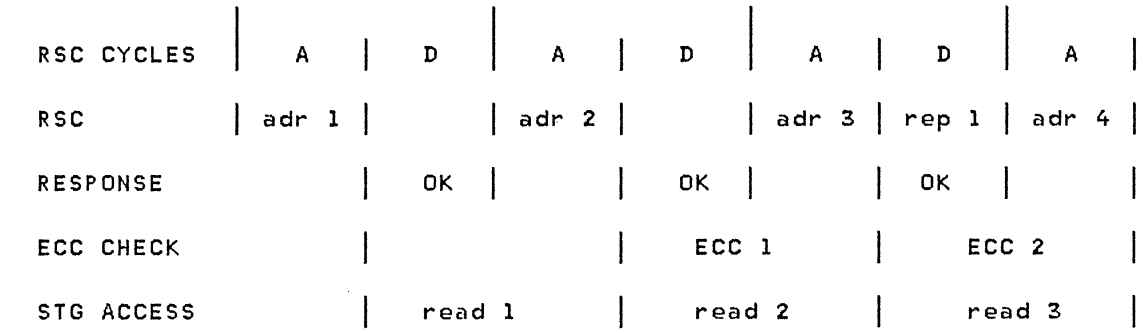


Figure 27. Storage Controller Timing With ECC

9.1 STORAGE PROTECT AND ADDRESS TRANSLATION OVERVIEW

ROMP supports both storage protect and address translation functions in the storage controller. These functions are enabled by control bits in the Interrupt Control Status (ICS) register. When bit 20 in the ICS is set to one, storage protect is enabled. When bit 22 is set to one, address translation is enabled. When either of these functions is enabled, ROMP is placed in an exact interrupt mode, which allows any instruction which causes an exception to be restarted after the exception has been handled. The state of ICS bits 20 and 22 is available on RSC lines DAL04 and DAL05, respectively. The actual storage protect or address translation hardware exists in the storage controller, and the particular implementation is system dependent. Required initialization of the storage protect or address translation hardware must be performed by system software before storage protect or address translation is enabled.

Assuming that protection or translation hardware is installed, the storage controller does the necessary checking, and provides the appropriate response when a reply is generated. In the case of a read access, storage will generate a reply to the original requestor. When no exception condition exists, data is placed on the addr/data bus, the tag ID of the requestor is placed on the tag bus, and +EXCEPTION is driven inactive. If an exception condition exists, the reply is generated in a similar manner, except +EXCEPTION is driven active and the data which is placed on the bus is not used, but must have good parity.

Whenever a write access is attempted and storage protect or address translation is enabled, a reply must be generated to the

original requestor, just as was done for a read access. This reply is required in order to determine whether or not the store operation caused an exception. The tag bus will contain the tag ID associated with the write address of the original requestor. The +EXCEPTION line will be driven inactive when no exception exists, and will be driven active when an exception does exist. In either case, the content of the data bus is unimportant, but must have good parity.

For systems which do not implement storage protection or virtual addressing, the EXCEPTION line is not required. However, EXCEPTION can still be used with replies to ROMP to report addressing exceptions. If ROMP receives a reply with +EXCEPTION active, and address translate and storage protect are disabled, then a program check occurs. PCS bits 25 (program check with unknown origin) and 30 (data address exception) are set. If the EXCEPTION line is not used, the input to ROMP must be tied low (inactive).

When storage accesses are generated by ROMP , the RSC line DAL06 will be a one (active) if the processor is in problem state and a zero (inactive) if in supervisor state. This bit may be used to enhance the storage protection scheme by providing different types of access authority, such as read/write, read only, or execute only.

9.2 STORAGE PROTECT

Storage protect provides controlled access to selected storage areas. Normally, each area (or block) is designated as execute only, read only, read/write, or no access allowed for a particular task. This requires associating the required access authority with each block of storage, and verifying that each storage request has the correct access authority.

The access authority is normally contained in a separate array in the storage controller that is accessed based on the address of the request. For a protect scheme with fixed size blocks of storage, the high-order address bits are used to access the protect array and the low-order address bits are ignored. The number of low-order address bits ignored is determined by the block size.

Information contained in the protect array is used by the storage controller to determine if each access is permitted. This information would normally indicate which RSC signals should be checked, and the state of these lines for a legal access.

Sufficient information is provided on the RSC to identify each type of storage operation. The RSC tag lines provide the ability to determine if a particular request is for a processor

instruction or data fetch, or whether the request is from another device on the RSC (bus converter, DMA controller, etc). The high-order byte of the RSC addr/data bus provides additional control information to determine if the request is a read or write, whether it is a storage access or a PIO access, and whether the processor is in problem or supervisor state. A detailed description of these signals is given in "RSC Signal Definitions" on page 97.

Typically, it might be desired to define a storage area that is read/write from the system control program (SCP) and is execute only for user programs. The storage controller can provide this checking by examining the state of the problem/supervisor line on the RSC (DAL06) and the tag lines to determine if a request is being made from problem or supervisor state, and whether or not it is an instruction fetch. If the processor is in supervisor state, any access would be permitted, and if in problem state, only instruction fetches would be permitted to the given storage area.

If a storage area is designated as read/write from the SCP and no access allowed from problem state, the storage controller would only need to examine the state of the problem/supervisor line to determine whether or not an access is permitted. In this case, the state of the tag lines is a don't care.

Note that separate storage areas can be designated for each type of device that can access storage. For example, selected storage areas can be designated as valid only for processor operations, at the same time other areas are valid only for DMA transfers. Figure 28 shows a typical storage address assignment.

A typical storage protect scheme requires access of control information from an array to determine if each access is allowed. Normally, this information is contained in a separate array, and is accessed based on the request address by the storage controller to determine if each access is legal. The access of this control information can normally occur in parallel with the data access. For write operations, this control information must be examined prior to the data access, in order to prevent illegal write operations from altering data in storage. For non-write accesses, the storage protect information can be examined after the data access is complete, to determine the state of the EXCEPTION line for the reply. Data read from storage can be returned to the processor if there is an exception, since ROMP does not use this data if EXCEPTION is active.

Once the exception indication is received by ROMP , execution of the current instruction is terminated, and any general purpose registers (GPRs) which were altered by partial execution of the instruction, and would prevent the instruction from being restartable, are restored. A program-check then occurs which causes appropriate bits in the program check status (PCS) register to be set based on the type of exception. Refer to "Program-Check Errors" on page 124 for a description of the PCS.

Control information in the storage protect array is normally initialized by the processor, before a particular task begins execution. Initialization defines each area that a particular task can access, and the type of access permitted (execute, read/write, etc). The storage protect array can be either memory mapped and initialized by store instructions, or can be defined in the I/O address space, and initialized by PIO write instructions.

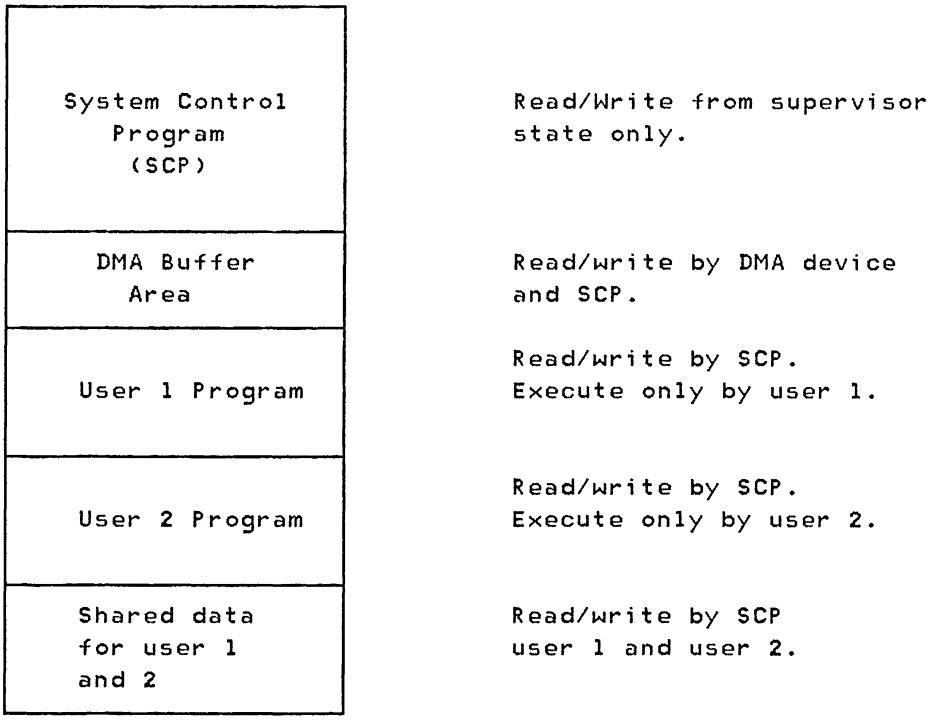


Figure 28. Typical Storage Protect Assignments

9.3 ADDRESS TRANSLATION

ROMP supports address translation functions which allow mapping of a virtual processor address to a physical storage address. The address translation hardware exists in the storage controller and the particular implementation is system dependent. This hardware

converts a virtual address from ROMP , or other system components, into a physical address which is used to access storage.

If an exception is detected by storage during execution of an instruction, the EXCEPTION line is brought active for the reply from storage, which causes a program-check condition to occur. Once the exception indication is received by ROMP , execution of the current instruction is terminated, and any general purpose registers (GPRs) which were altered by partial execution of the instruction, and would prevent the instruction from being restartable, are restored. A program-check then occurs which causes appropriate bits in the program check status (PCS) register to be set based on the type of exception. Refer to "Program-Check Status" on page 125 for a description of the PCS.

An exception handler can then be executed to determine the type of exception, and can take the appropriate action to eliminate the exception. This routine can examine bits in the PCS to determine if the exception was the result of an instruction fetch or data reference. If the exception resulted from an instruction fetch, the IAR in the old program check (PC) PSW will contain the address of the instruction causing the exception. In this case, the instruction address is also the storage address causing the exception. If the exception resulted from a load or store operation, the IAR in the old PC PSW will contain the address of the load or store instruction, not the storage address causing the exception. Hardware can be provided in the storage controller to maintain the storage address causing the exception, or software can determine the address by examining the instruction pointed to by the IAR in the old PSW. If hardware is provided in the storage controller, the storage address causing the exception can be made available to the processor via a PIO register in the storage controller.

A separate pipelined stage can be added in the storage controller to perform address translation. This allows storage array access to be overlapped with the next address translation. A design such as this prevents the address translation time from being directly added to the storage access time. Figure 29 shows how address translation and storage array access can be overlapped.

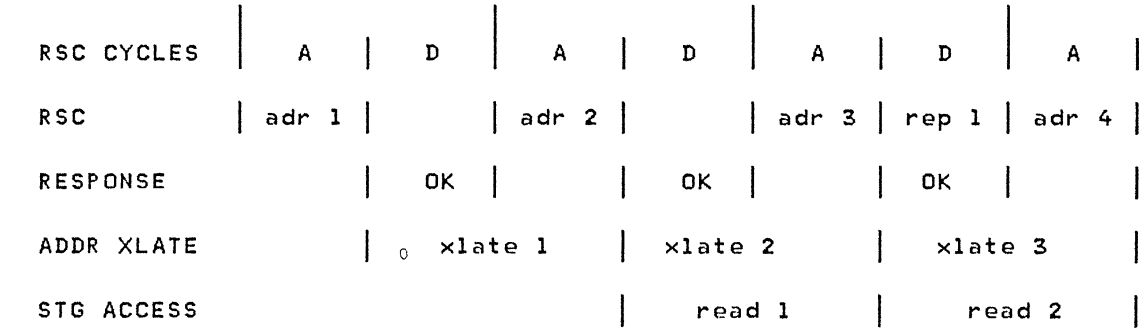


Figure 29. Storage Controller Timing With Address Translation

10.0 PROCESSOR SUPPORT FUNCTIONS

10.1 FRONT PANEL SUPPORT

ROMP provides internal hardware to support basic front panel functions in a normal system environment. This hardware supports processor reset and IPL operations and provides an output to indicate processor failure. Reset and IPL can be performed automatically when power is applied to the system, or can be performed manually via front panel switches. Processor failure is indicated by an I/O pin (-FAIL) which can be connected to a front panel failure indicator. "Internal Diagnostics" on page 121 describes operation of the failure signal.

10.2 SUPPORT PROCESSOR FACILITIES

ROMP is an LSSD design processor which allows contents of the internal latches to be examined and altered by use of the LSSD scan feature. In addition, internal support logic is provided for sync and stop-on-address functions.

Internal address compare logic consists of a 32-bit address-compare register, address source select latch, and a stop-on-address enable latch. The address-compare register is used to hold the desired sync or stop address. The source select latch selects either an IAR address or microstore address. If the stop-on-address latch is set, the processor is forced into a stopped state when an address compare occurs. An I/O pin (-SYNC) is provided for use as a trigger source when using the sync function to examine RSC or other processor activity.

For system or program debug, a support processor such as a PC, PT-2, or a Series/1 can be connected to a ROMP system to allow examination of internal ROMP registers. The support processor would use the LSSD scan-in and scan-out capability to examine and alter internal processor registers and control latches. In addition, the address compare logic can be controlled by the support processor for sync or stop-on-address functions. A detailed description of the LSSD scan-strings and a suggested support processor interface can be found in documents listed in "ROMP System Hardware References" on page 188.

The particular functions provided by the support processor are defined by the program written for the support processor, but should include the following functions:

1. Reset. It should be possible to reset ROMP from the support processor. Reset places the ROMP system in a known state. This function can be implemented by ORing an output from the support processor with the ROMP system reset signal.

2. IPL. After ROMP has been reset, it begins register initialization. At the completion of register initialization, ROMP activates -IPL READY. This signal can be detected by the support processor, which can take one of several courses of action. The support processor can place ROMP in the stopped state, which allows the contents of scan strings, registers, and main storage to be displayed and altered as explained below. Also, the support processor can emulate an IPL device, loading main storage and activating -IPL COMPLETE when the load is complete. Finally, the support processor can allow the ROMP system to complete IPL as it normally would.

3. Start/Stop/Step. These functions allow operation of the ROMP processor to be controlled. Start would enable processor execution to begin with the current IAR value. Stop halts processor execution. Step allows single stepping of instruction execution while the processor is in the stopped state. These functions can be implemented by interfacing the support processor with the ROMP clock generation circuitry.

4. Address Compare. This function allows entry of a desired compare address, selection of compare on IAR value or microstore address, and enable of stop-on-address. The support processor would load the compare address register with the desired address via the LSSD scan-in input. IAR or microstore address is selected by scanning the desired value into the select latch and stop-on-address is enabled by scanning the stop latch to an enabled state.

5. Display/Alter GPRs and SCRs. This function allows the support processor to read and write any GPR or SCR. These functions would be implemented by manipulation of the appropriate control and data registers via the scan-in and scan-out paths.

6. Display/Alter Main Storage. This function allows the support processor to load, read and modify main storage locations. The function can be implemented by scanning the desired address and data into the serial interface provided in the storage controller.

7. Display/Alter ROMP Data and Control Registers. This function allows internal data and control registers to be examined and altered. This function can be implemented by using the support processor to scan-out the data and control latch contents for display. The scanned-out can then be altered, if desired, and scanned-in to set a control or data latch to a desired value.

11.0 PERFORMANCE

ROMP performance can be evaluated by computing instruction execution times based on the figures shown in Table 13.1 and by including any hold-off time for instruction or data fetches. A hold-off condition in the processor will occur whenever a successful branch is taken and the instruction buffer must start fetching a new instruction stream from storage, or whenever an instruction references a register which has not yet been loaded as the result of a load instruction. Any processor activity such as interrupts or system timer service must be included in performance computations since they require execution of microcode routines. Performance limitations due to the RSC bandwidth must be evaluated where RSC utilization is high.

The term "storage access time" used in this section is the amount of time required by the storage controller to reply to an instruction or data fetch. This time includes driver delays from the storage controller to the storage array, the array access time, receiver delays from the array to the storage controller, and any ECC delays, if present. The storage access time is always an integer multiple of the processor cycle time due to the synchronous nature of the ROMP Storage Channel. See "Selection of Processor Cycle Time" on page 150 for further information on storage access time versus processor cycle time.

The term "RSC cycle time" used in this section is the packet time on the RSC. This time is always one-half of the processor cycle time. For example, a 300 nsec processor cycle results in a 150 nsec RSC cycle time, and a 200 nsec processor cycle results in a 100 nsec RSC cycle time.

For the purpose of discussion in this section, the ROMP cycle time is assumed to be 300 nsec. The actual cycle time is system dependent.

This section is intended to provide an overview of the various factors involved in ROMP performance. However, due to the large number of possible system configurations, it is difficult to obtain detailed performance estimates for a given system without use of a simulator. Detailed performance analysis for a given system configuration with the anticipated instruction mix should be done using the simulator described in "RTIMER Simulator" on page 188.

11.1 BRANCH HOLD-OFF

The ROMP design utilizes an instruction prefetch buffer to reduce apparent instruction fetch time. In normal operation, the

instruction buffer prefetches instructions prior to their execution so the effective fetch time during execution is zero. However, whenever a branch instruction is successful, the instruction buffer must fetch a new instruction stream and the processor is forced into a hold-off state until a new instruction is available for execution. The length of time the processor is in a hold-off state depends on the RSC cycle time and on the storage access time. Figure 30 shows the timing for instruction fetches.

The hold-off time for the start of a new instruction stream to begin execution is given by the following equation:

Twait= 2*RSC Cycle Time + Storage Access Time + 1 Processor Cycle Time

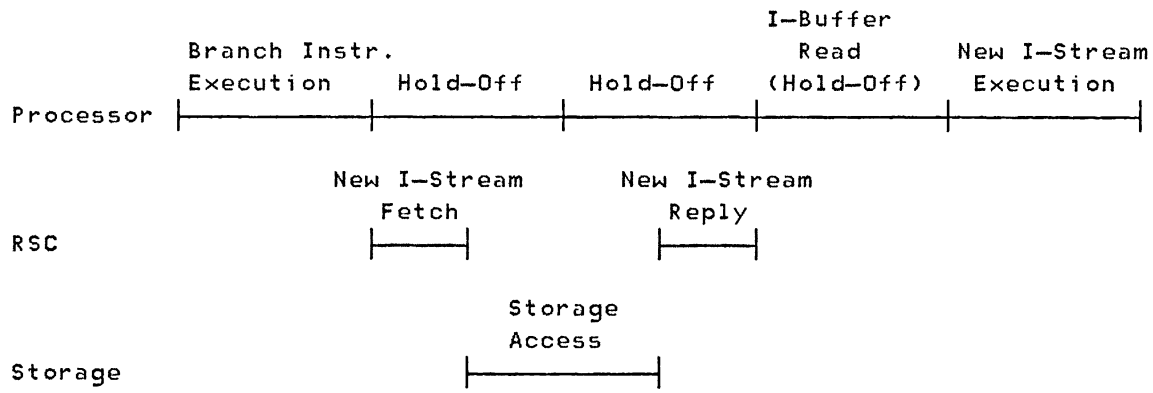


Figure 30. Fetch Timing For New Instruction Stream Following A Successful Branch

For an RSC and ROMP design with a 150nsec RSC cycle time and a 300nsec ROMP cycle time, the hold-off time is given by:

Twait = 600nsec + Storage Access Time

For a typical system with 300nsec access time storage, the wait time would be 900nsec or three processor cycles. This hold-off time must be added to the execution time for each branch, if it is successful.

11.2 BRANCH AND EXECUTE HOLD-OFF

The ROMP instruction set includes branch and execute instructions which allow a subject instruction to be executed while the target instruction is being fetched. In this case, the time until execution of the target instruction begins is the greater of the time required for execution of the subject instruction or the time required for the new instruction stream to become available for execution. If execution of the subject instruction is complete before the target instruction is available for execution, the processor enters a hold-off state until the target instruction is available. Subject instruction execution times can be computed from Table 13.1 and hold-off time for the target instruction is defined in "Branch Hold-off" on page 144.

11.3 LOAD INSTRUCTION HOLD-OFF

The ROMP design allows instruction execution to continue in parallel with the loading of a register, provided the subsequent instructions do not reference an unloaded register when storage protect and address translation are disabled. Hold-offs that occur as a result of storage protect and address translation are described in "Storage Protect And Address Translation Hold-Off" Execution of a load instruction causes a tag to be allocated for the given register and a request to storage for the given data. Instructions following the load instruction can be executed while the storage access and writing of the registers occur, provided the subsequent instructions do not reference an unloaded register. Figure 31 shows the timing for a load register operation.

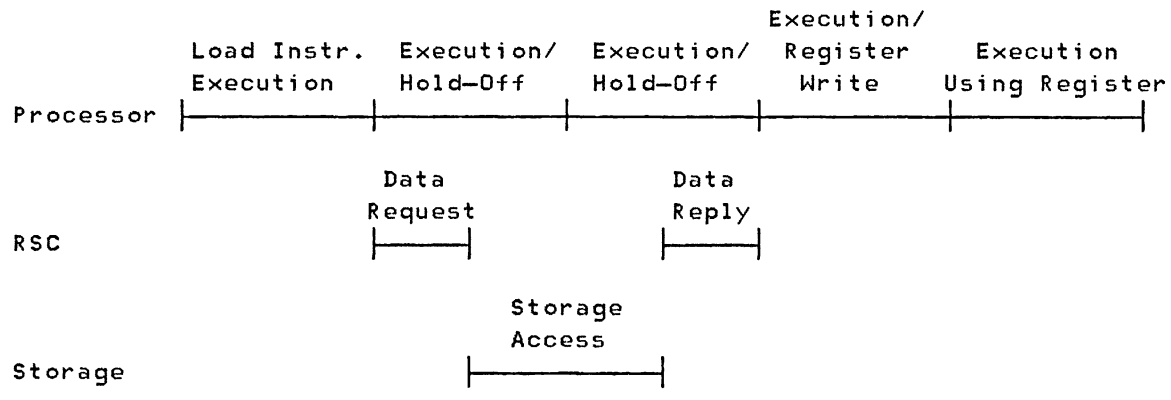


Figure 31. Load Instruction Timing

The actual time required for completion of a load instruction is given by the following equation:

Tload = 2* RSC Cycle Time + Storage Access Time + 1 Processor Cycle

For an RSC and ROMP design with a 150 nsec RSC cycle time and a 300nsec ROMP cycle time, the load time is:

Tload = 600 nsec + Storage Access Time

For a typical system with 300 nsec access time storage, the load time would be 900 nsec or three processor cycles. Instruction execution can continue in parallel with the load operation provided the subsequent instructions do not reference an unloaded register. If a subsequent instruction references an unloaded register, the processor is forced into a hold-off state until the load operation is completed.

11.4 I/O READ HOLD-OFF

Execution of an I/O read instruction causes a tag to be allocated for the given register and a RSC command to the I/O device to be generated. Hold-off conditions previously described for a load register operation also apply to I/O read operations. Previous timing diagrams and equations for determining register load times can be used for I/O read operations by substituting the I/O device response time for storage access time. No additional hold-offs occur when storage protect or address translation is enabled, since all I/O addresses are real.

11.5 STORAGE PROTECT AND ADDRESS TRANSLATION HOLD-OFF

When storage protect or address translation is enabled, additional hold-offs occur after each load or store operation. In this mode, subsequent instructions are not executed until the storage controller responds to the load or store operation. Operation of the storage controller in storage protect or address translate mode is described in "Storage Controller Functions" on page 134. Figure 32 shows the timing for load and store operations when storage protect or address translation is enabled. Note that the storage controller must also respond to write operations in this mode. Data returned by the storage controller for a write operation is not used by the processor; only the tag information

and state of the EXCEPTION line is used to determine the outcome of the write operation.

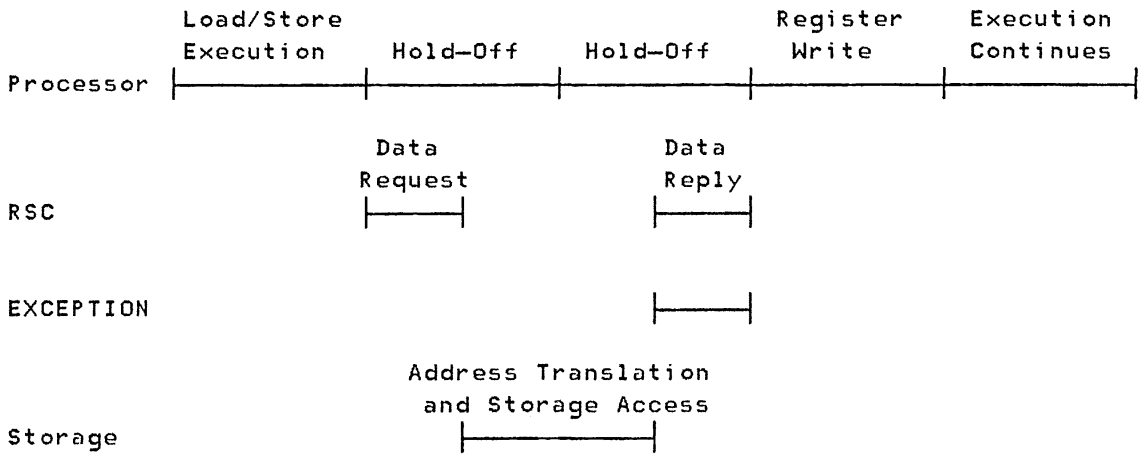


Figure 32. Load and Store Instruction Timing With Storage Protect or Address Translation Enabled

Performance degradation occurs since subsequent instruction execution is halted during a load operation regardless of whether or not subsequent instructions reference the register. Hold-offs now also occur after write operations until the the status of the write operation is reported by the storage controller. No additional hold-offs occur after I/O reads, since all I/O addresses are real. Prefetching occurs as it normally would, with no additional hold-offs between prefetches.

Performance can be evaluated by including these additional hold-off times and the increased storage access time due to address translation in performance calculations. Note that as the storage array access time increases, the overhead of storage protect or address translation becomes a smaller portion of the overall storage access time, hence there is a smaller percentage degradation in system performance.

11.6 TAG HOLD-OFFS

ROMP provides two register tags which are allocated for all storage read operations, all I/O reads, and storage write

operations when address translation or storage protection is enabled. This permits two of these operations to be to be in progress at any time. If an attempt is made to execute an instruction and there are already two allocated tags, ROMP is forced into a hold-off state until a tag becomes available. A tag becomes available in the register write cycle (See Figure 31).

For example, if an instruction sequence includes three successive load operations, the first two loads are executed with no hold-offs, and the third must wait for the first load to complete. In a system with 300 nsec storage, there would be a hold-off of two cycles before the third load instruction would execute. The two cycle hold-off occurs since a tag is not available until three cycles after execution of a load instruction, and only one cycle has elapsed since execution of the first load operation.

11.7 INTERRUPTS

Interrupts cause execution of a microcode routine which performs a program status exchange. The execution time for this routine must be included when performance calculations are made involving interrupts. Table 13.1 lists the number of cycles required for execution of the microcode interrupt routine.

11.8 SYSTEM TIMER

The system timer is serviced by a microcode routine which updates and sets the required timer status bits for each clock input to the timer. The execution time for this routine must be included when performance calculations are made involving system timer operation. Table 13.1 lists the number of cycles required for execution of the microcode system timer routine.

11.9 BUS CAPACITY

A limiting factor in processor performance is the bandwidth of the storage channel. The RSC is designed to provide sufficient bandwidth for both processor and DMA activity without limiting processor performance. However, if there are high speed DMA devices on the RSC; a point can be reached where DMA activity saturates the RSC which will result in degraded processor performance.

An RSC design with a 150 nsec bus cycle time provides a maximum bus bandwidth of 13.3 M bytes/sec (4 bytes every 300 nsec) which can be achieved with 300 nsec cycle time storage. A typical execution rate of two MIPS requires approximately a 8 M byte/sec data rate for the processor which leaves approximately 5 M bytes/sec for other RSC devices As RSC device activity increases, processor performance will be degraded slightly due to interference between the processor and devices. As the total RSC utilization approaches 100%, processor performance will degrade so that close to the 13.3 M byte/sec RSC capacity is maintained. Actual bus capacity required for a given program can be estimated as described in the following section.

11.10 SELECTION OF PROCESSOR CYCLE TIME

The processor cycle time can be adjusted in order to maximize the available storage bandwidth. Since the RSC is a synchronous channel, storage accesses always require an integer number of processor cycles. If the storage array access time is not an integer multiple of the processor cycle, there will be a delay between the time that data is available in the storage controller, and when the data can be returned to ROMP via the RSC. This delay can be eliminated by selecting the processor cycle time so that storage access time is an integer multiple of the processor cycle time. In many systems, the overall system performance will be limited by the available storage bandwidth, and not by the processor cycle time. In this case, the overall system performance can actually be improved by increasing the processor cycle time, so that the storage access time is an integer multiple of the processor cycle time.

Consider a system with a 250 nsec processor cycle time and 600 nsec access time storage. Since the storage controller can only reply every every 250 nsec, and data is not good until 600 nsec, the actual access time will be 750 nsec. This results in a storage bandwidth of 5.33 M bytes/sec (4 bytes every 750 nsec). If the processor cycle time is increased to 300 nsec, the access time will become 600 nsec, giving a bandwidth of 6.67 M bytes/sec. In this case, increasing the processor cycle time by 50 nsec has

increased the available storage bandwidth by approximately 25 percent.

11.11 PROGRAM PERFORMANCE

The required execution time for a given program can be estimated by summing the number of cycles required for each instruction executed by the program and by adding this number to any hold-off condition times that occur during program execution. The hold-off times are determined by events described in "Branch Hold-off" on page 144 through "Bus Capacity" on page 150. The number of cycles required for execution of each instruction is listed in Table 13.1.

Once execution time is determined, it is then possible to determine the RSC utilization for the program. This is done by computing the total number of bytes during program execution and dividing by the execution time of the program. The number of bytes required during execution of the program is the sum of the number of instruction bytes fetched, the number of data bytes referenced, and the number of bytes wasted by the instruction prefetch buffer for successful branches. The number of instruction bytes fetched can be computed by multiplying the number of instructions executed by the average instruction length. For typical programs, the average instruction length is approximately 2.4 bytes. The number of data references includes all load and store operations and each operation requires a 4 byte transfer. Since the instruction prefetch buffer contains instructions ahead of the execution stream, successful branches will result in some of the instructions in the buffer being wasted. These instructions have already been fetched from storage and must be included in bus performance computations. The current ROMP design includes 16 byte instruction prefetch buffer and performance estimates generally assume approximately 8 bytes are wasted for each successful branch.

Consider evaluation of a program with the following characteristics:

```
400   Instructions executed
600   processor cycles required for program execution
 60   hold-off cycles (successful branches, data references, etc)
100   data references (loads and stores)
 30   successful branches
```

For a 300 nsec processor cycle, this program would execute in (600+60 cycles) X 300 nsec/cycle or approximately 200 usec. The MIP rate would be 400 instructions divided by 200 usec or 2 MIPS. Note that any processor cycles required for interrupt, or system

timer servicing would be added to the 600 processor cycles required for program execution.

Bus utilization would be computed as follows:

400	instructions *2.4 bytes/instruction	960 Bytes
100	data references *4 bytes/reference	400
30	successful branches *8 bytes/branch wasted	240
	Total	1600 Bytes

$$\text{Required bandwidth:} \quad \frac{1600 \text{ Bytes}}{200 \text{ usec}} = 8.0 \text{ MBytes/Sec}$$

A minimum storage bandwidth of 8 M bytes/sec is required for this program to execute in the previously computed 200 usec. As storage access time is increased, execution time for the program will increase in a nonlinear manner, and total storage utilization will approach 100%.

In cases where the actual storage capacity is less than that required to support a given execution rate, the actual execution rate can be estimated by assuming full utilization of the available storage bandwidth. Using data from the above example and assuming non interleaved SUNSET memory with a 1200 nsec cycle time, estimated program performance can be computed as follows:

$$\text{Storage Capacity:} \quad \frac{4 \text{ Bytes}}{1200 \text{ nsec}} = 3.33 \text{ M Bytes/Sec}$$

$$\text{Estimated Execution Time:} \quad \frac{1600 \text{ Bytes}}{3.33 \text{ M Bytes/Sec}} = 480 \text{ usec}$$

$$\text{MIP Rate:} \quad \frac{400 \text{ Instructions}}{480 \text{ usec}} = 0.83 \text{ MIPS}$$

Note that the actual execution rate will be slightly lower than the computed value since additional hold-off cycles will occur for successful branches and load operations due to the longer storage access time. Also, note that as storage utilization approaches

100%, less instruction prefetching will be possible which will result in fewer bytes being wasted for successful branches.

11.12 PERFORMANCE MEASUREMENT

ROMP provides an I/O pin (-INST CMPLT) to indicate completion of instruction execution. A transition from inactive to active occurs when execution of an instruction is completed. The frequency of the signal at this pin represents instruction execution rate. For example, a frequency of 500 Khz represents 0.5 MIPS, 1 MHz represents 1 MIP, and 2 MHz represents 2 MIPS.

TABLE 13.1

INSTRUCTION EXECUTION TIMES

MNEMONIC	OP-CODE	INSTRUCTION	Execution Cycles
A	E1	ADD	1
ABS	E0	ABSOLUTE	2
AE	F1	ADD EXTENDED	1
AEI	D1	ADD EXTENDED IMMEDIATE	1
AI	C1	ADD IMMEDIATE	1
AIS	90	ADD IMMEDIATE SHORT	1
BALA	8A	BRANCH AND LINK ABSOLUTE	3 +1 Storage
BALAX	8B	BRANCH AND LINK ABSOLUTE WITH EXECUTE	3 +1 Storage
BALI	8C	BRANCH AND LINK IMMEDIATE	3 +1 Storage
BALIX	8D	BRANCH AND LINK IMMEDIATE WITH EXECUTE	3 +1 Storage
BALR	EC	BRANCH AND LINK	3 +1 Storage
BALRX	ED	BRANCH AND LINK WITH EXECUTE	3 +1 Storage
BB	8E	BRANCH ON CONDITION BIT IMMEDIATE	
		Unsuccessful	1
		Successful	3 +1 Storage
BBR	EE	BRANCH ON CONDITION BIT	
		Unsuccessful	1
		Successful	3 +1 Storage
BBRX	EF	BRANCH ON CONDITION BIT WITH EXECUTE	
		Unsuccessful	1
		Successful	3 +1 Storage
BBX	8F	BRANCH ON CONDITION BIT IMMEDIATE WITH EXECUTE	
		Unsuccessful	1
		Successful	3 +1 Storage
BNB	88	BRANCH ON NOT CONDITION BIT IMMEDIATE	
		Unsuccessful	1
		Successful	3 +1 Storage
BNBR	E8	BRANCH ON NOT CONDITION BIT	
		Unsuccessful	1
		Successful	3 +1 Storage
BNBRX	E9	BRANCH ON NOT CONDITION BIT WITH EXECUTE	
		Unsuccessful	1
		Successful	3 +1 Storage
BNBX	89	BRANCH ON NOT CONDITION BIT IMMEDIATE WITH EXECUTE	
		Unsuccessful	1
		Successful	3 +1 Storage
C	B4	COMPARE	1
CAL	C8	COMPUTE ADDRESS LOWER	1
CAL16	C8	COMPUTE ADDRESS LOWER HALF 16-BIT	1
CAS	6	COMPUTE ADDRESS SHORT	1
CAU	D8	COMPUTE ADDRESS UPPER	1
CA16	F3	COMPUTE ADDRESS 16-BIT	1

CI	D4	COMPARE IMMEDIATE	1
CIS	94	COMPARE IMMEDIATE SHORT	1
CL	B3	COMPARE LOGICAL	1
CLI	D3	COMPARE LOGICAL IMMEDIATE	1
CLRBL	99	CLEAR BIT LOWER HALF	1
CLRBU	98	CLEAR BIT UPPER HALF	1
CLRSB	95	CLEAR SCR BIT	4
CLZ	F5	COUNT LEADING ZEROS	1
D	B6	DIVIDE STEP	3
DEC	93	DECREMENT	1
EXTS	B1	EXTEND SIGN	1
INC	91	INCREMENT	1
IOR	CB	INPUT/OUTPUT READ	1
IOW	DB	INPUT/OUTPUT WRITE	2
JB	08-0F	JUMP ON CONDITION BIT	
		Unsuccessful	1
		Successful	3 +1 Storage
JNB	00-07	JUMP ON NOT CONDITION BIT	
		Unsuccessful	1
		Successful	3 +1 Storage
L	CD	LOAD	1
LC	CE	LOAD CHARACTER	1
LCS	4	LOAD CHARACTER SHORT	1
LH	DA	LOAD HALF	1
LHA	CA	LOAD HALF ALGEBRAIC	1
LHAS	5	LOAD HALF ALGEBRAIC SHORT	1
LHS	EB	LOAD HALF SHORT	1
LIS	A4	LOAD IMMEDIATE SHORT	1
LM	C9	LOAD MULTIPLE	See Note 2
LPS	DO	LOAD PROGRAM STATUS	6+ 4 Storage
LS	7	LOAD SHORT	1
M	E6	MULTIPLY STEP	4
MC03	F9	MOVE CHARACTER ZERO FROM THREE	1
MC13	FA	MOVE CHARACTER ONE FROM THREE	1
MC23	FB	MOVE CHARACTER TWO FROM THREE	1
MC33	FC	MOVE CHARACTER THREE FROM THREE	1
MC30	FD	MOVE CHARACTER THREE FROM ZERO	1
MC31	FE	MOVE CHARACTER THREE FROM ONE	1
MC32	FF	MOVE CHARACTER THREE FROM TWO	1
MFS	96	MOVE FROM SCR	2
MFTB	BC	MOVE FROM TEST BIT	1
MFTBIL	9D	MOVE FROM TEST BIT IMMEDIATE LOWER HALF	1
MFTBIU	9C	MOVE FROM TEST BIT IMMEDIATE UPPER HALF	1
MTS	B5	MOVE TO SCR	3
MTTB	BF	MOVE TO TEST BIT	1
MTTBIL	9F	MOVE TO TEST BIT IMMEDIATE LOWER HALF	1
MTTBIU	9E	MOVE TO TEST BIT IMMEDIATE UPPER HALF	1
N	E5	AND	1
NILO	C6	AND IMMEDIATE LOWER HALF EXTENDED ONES	1

NILZ	C5	AND IMMEDIATE LOWER HALF EXTENDED ZEROS	1
NIUO	D6	AND IMMEDIATE UPPER HALF EXTENDED ONES	1
NIUZ	D5	AND IMMEDIATE UPPER HALF EXTENDED ZEROS	1
O	E3	OR	1
OIL	C4	OR IMMEDIATE LOWER HALF	1
OIU	C3	OR IMMEDIATE UPPER HALF	1
ONEC	F4	ONE'S COMPLEMENT	1
S	E2	SUBTRACT	1
SAR	B0	SHIFT ALGEBRAIC RIGHT	1
SARI	A0	SHIFT ALGEBRAIC RIGHT IMMEDIATE	1
SARI16	A1	SHIFT ALGEBRAIC RIGHT IMMEDIATE PLUS SIXTEEN	1
SE	F2	SUBTRACT EXTENDED	1
SETBL	9B	SET BIT LOWER HALF	1
SETBU	9A	SET BIT UPPER HALF	1
SETSB	97	SET SCR BIT	4
SF	B2	SUBTRACT FROM	1
SFI	D2	SUBTRACT FROM IMMEDIATE	1
SIS	92	SUBTRACT FROM SHORT	1
SL	BA	SHIFT LEFT	1
SLI	AA	SHIFT LEFT IMMEDIATE	1
SLI16	AB	SHIFT LEFT IMMEDIATE PLUS SIXTEEN	1
SLP	BB	SHIFT LEFT PAIRED	1
SLPI	AE	SHIFT LEFT PAIRED IMMEDIATE	1
SLPI16	AF	SHIFT LEFT PAIRED IMMEDIATE PLUS SIXTEEN	1
SR	B8	SHIFT RIGHT	1
SRI	A8	SHIFT RIGHT IMMEDIATE	1
SRI16	A9	SHIFT RIGHT IMMEDIATE PLUS SIXTEEN	1
SRP	B9	SHIFT RIGHT PAIRED	1
SRPI	AC	SHIFT RIGHT PAIRED IMMEDIATE	1
SRPI16	AD	SHIFT RIGHT PAIRED IMMEDIATE PLUS SIXTEEN	1
ST	DD	STORE	2
STC	DE	STORE CHARACTER	2
STCS	1	STORE CHARACTER SHORT	2
STH	DC	STORE HALF	2
STHS	2	STORE HALF SHORT	2
STM	D9	STORE MULTIPLE	See Note 2
STS	3	STORE SHORT	2
SVC	C0	SUPERVISOR CALL	11 +2 Storage
TGTE	BD	TRAP IF REGISTER GREATER THAN OR EQUAL	
		Unsuccessful	2
		Successful	10 +2 Storage
TI	CC	TRAP ON CONDITION IMMEDIATE	
		Unsuccessful	2
		Successful	10 +2 Storage
TLT	BE	TRAP IF REGISTER LESS THAN	
		Unsuccessful	2
		Successful	10 +2 Storage

TSH	CF	TEST AND SET HALF	1
TWOC	E4	TWOS COMPLEMENT	1
WAIT	F0	WAIT	1
X	E7	EXCLUSIVE OR	1
XIL	C7	EXCLUSIVE OR IMMEDIATE LOWER HALF	1
XIU	D7	EXCLUSIVE OR IMMEDIATE UPPER HALF	1
INTERRUPT			8 +2 Storage
SYSTEM TIMER UPDATE 3			

Notes:

1. Storage is storage access time as defined in "Performance" on page 144.

2. Execution cycles for Load Multiple (LM) and Store Multiple (STM) can be computed by the following equations. R is the number of registers to be loaded or stored, Tacc is the storage access time, Tcyc is the storage cycle time, and Tprocessor is the processor cycle time. Tacc and Tcyc are expressed as an integer multiple of the processor cycle. For example, a 250 nsec processor cycle with a storage access time of 500 nsec and a storage cycle time of 900 nsec, Tacc equals 2 and Tcyc equals 4. See "Performance" on page 144 for a definition of storage access time.

   Load Multiple execution cycles for 2-way interleaved storage where Tacc is any value, or fast non-interleaved storage where Tacc <= Tprocessor is given by the following equation. Tcyc is assumed to be less than or equal to two times Tacc. If Tacc < Tprocessor, use 1 for Tacc in the following equation.

   Execution cycles = 3 + R + (Tacc * K1)

   K1 equals Integer[(R-1)/2] if address translation and storage protect are disabled. K1 equals Integer[(R+1)/2] if address translation or storage protect is enabled.

   Load Multiple execution cycles with slow non-interleaved storage where Tacc > Tprocessor :

   Execution cycles = 3 + R + (Tacc * K2)

   K2 equals R-1 if address translation and storage protect are disabled. K2 equals R+1 if address translation or storage protect is enabled.

   Store Multiple execution cycles for non-interleaved storage where Tcyc <= 2*Tprocessor, or 2-way interleaved storage where Tcyc <= 4*Tprocessor:

   Execution cycles = K3 + (2 * R)

K3 equals 1 if address translation and storage protect are disabled. K3 equals 2 if address translation or storage protect is enabled.

Store Multiple execution cycles for non-interleaved storage where Tcyc > 2*Tprocessor:

Execution cycles = K3 + (2 * R) If R <= 2

Execution cycles = K3 + (2 * R) + (Tcyc * (R - 2)) If R > 2

Store Multiple execution cycles for 2-way interleaved storage where Tcyc > 4*Tprocessor:

Execution cycles = K3 + (2 * R) If R <= 3

Execution cycles = K3 + (2 * R) + ((Tcyc * (R - 4)) If R odd and R > 3

Execution cycles = K3 + (2 * R) + ((Tcyc - 2) * (R - 3)) If R even and R > 3

12.0 HARDWARE DESCRIPTION

This section describes the hardware interfaces to the ROMP chip, and provides a brief overview of the various interfaces. References are provided to previous sections which describe these interfaces in detail. A detailed specification of the voltage level and timing requirements for each interface is provided in the ROMP Engineering Specification.

12.1 ROMP CHIP INTERFACES

There are five groups of interface signals to the ROMP chip. They are the ROMP Storage Channel (RSC), the system clocks, power, the interrupt inputs, and ROMP controls. Each of these interface signals (as shown in Figure 33), is described in "ROMP Storage Channel" on page 160 through "ROMP Controls" on page 168.

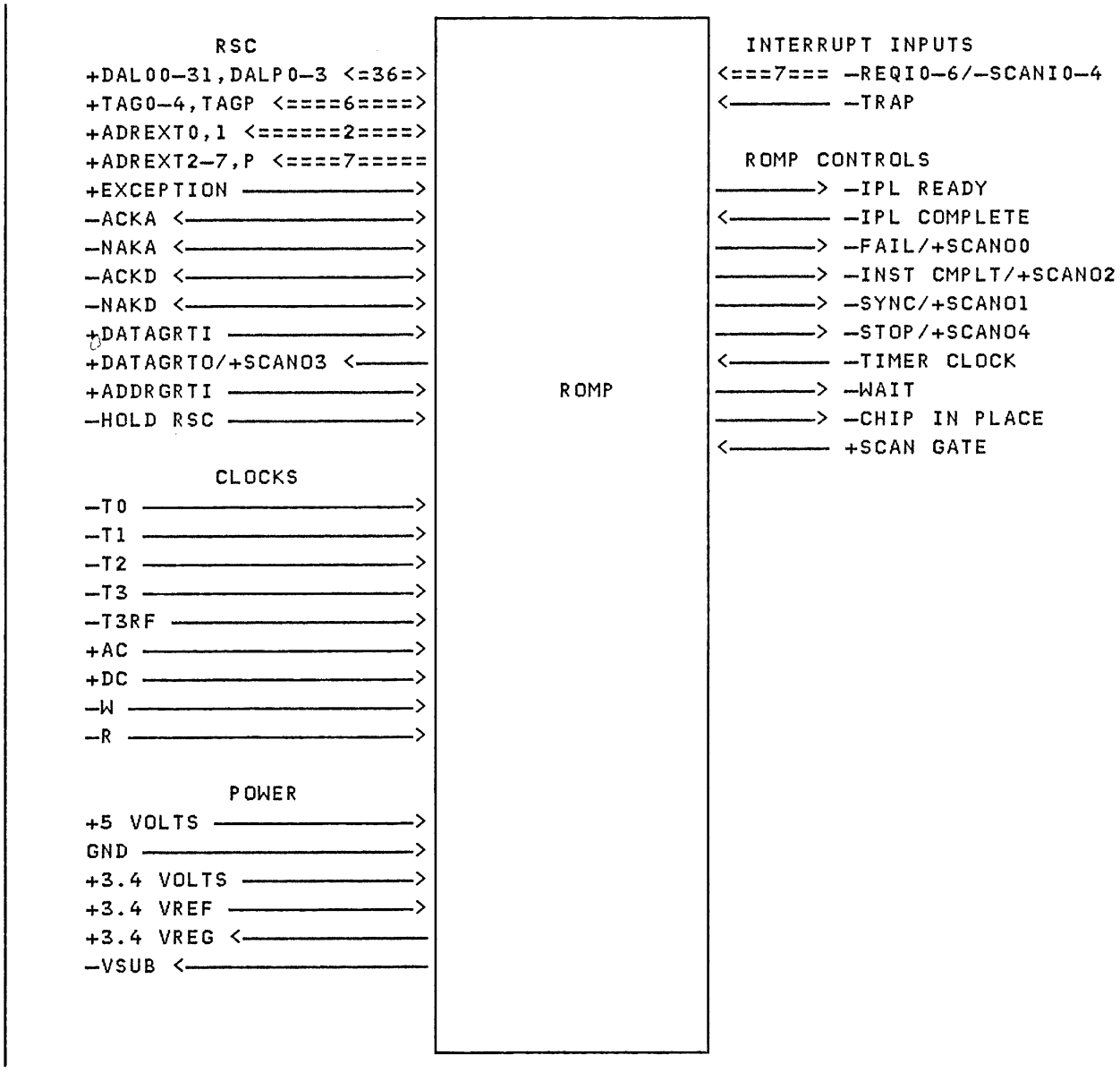


Figure 33. ROMP Module Signals

12.1.1 ROMP Storage Channel

This section is intended to provide a brief overview of the RSC, and is not intended to provide a detailed explanation of the RSC operation. For a detailed description of the RSC, refer to "ROMP Storage Channel" on page 92

The ROMP Storage Channel (RSC) is a high-bandwidth synchronous bus designed to interconnect a ROMP , a storage unit, and one or more

RSC devices. It supports a 32-bit data transfer and a 32-bit address (The basic RSC supports a 24-bit address, with an address extension bus for 32-bit addressing). Read operations on the RSC consist of two uncoupled transfers, a request and a reply, which allows multiple operations to overlap. This feature, combined with several features in the ROMP data flow, allows high processor performance with relatively slow storage through interleaving techniques.

The main elements of the RSC are a 32-bit (plus 4 parity) multiplexed Data/Address bus and a 5-bit (plus 1 parity) Tag bus. The Data/Address bus contains either 32-bits of data or a 24-bit address plus a byte of control information. The Tag bus contains codes which link replies to requests. An Address Extension bus provides 8 high-order address bits which extend the address to 32 bits. In addition, there are several miscellaneous handshaking, control, and clock lines.

The RSC runs synchronously with ROMP , with two RSC cycles per ROMP cycle. The first RSC cycle is always used to transmit addresses, and the second is used for data. There are three types of RSC transfers, called packets. A read request is a single address cycle, a write request is an address cycle plus the following data cycle, and a reply is a single data cycle. These requests are shown in Figure 34.

The RSC architecture allows any device to assume control of the RSC and issue requests. In a typical system, ROMP would issue requests to storage or other RSC devices, and RSC devices would issue requests to storage and each other.

Control of the RSC is determined by two arbitration systems, one for requests (Address Grant) and one for replies (Data Grant). Arbitration is for a period of two RSC cycles, with reply and request arbitration being overlapped in time with each other, and also with bus transfers. The arbitration systems are defined to be daisy-chained, but it is possible to implement a radial arbiter.

12.1.1.1 RSC Address and Data Bus

The RSC provides a 32-bit multiplexed address and data bus with byte parity. During an address cycle, +DAL00 through +DAL07 provide control information, and +DAL08 through +DAL31 provide a 24-bit address. Parity bits +DALP0 through +DALP3 provide odd parity for the control byte (+DAL00 through +DAL07) and the three address bytes (+DAL08 through +DAL15, +DAL16 through +DAL23, and +DAL24 through +DAL31). During a data cycle, +DAL00 through +DAL31 provide 32 bits of data. Parity bits +DALP0 through +DALP3

	<-------ROMP CYCLE-------->	
	<-RSC CYCLE->	<-RSC CYCLE->
Read Request	Address	
Write Request	Address	Data
Reply		Data

Figure 34. RSC Transfers

provide odd parity for for the four data bytes. See "Address/Data Bus Definition" on page 97 for more information.

12.1.1.2 RSC Tag Bus

The RSC provides a 5-bit tag bus which identifies the source of requests for transfers on the RSC. This 5-bit bus consists of +TAG0 through +TAG4. Odd parity is provided on the tag bus by +TAGP. See "Tag Bus Definition" on page 98 for more information.

12.1.1.3 RSC Address Extension Bus

The address extension bus provides an additional eight address bits for ROMP systems implementing 32-bit addressing. The high-order address byte of the 32-bit address, is placed on the address extension bus +ADREXT0 through +ADREXT7 during an address cycle. The three low-order address bytes are on the RSC address and data bus (+DAL08 through +DAL31) during an address cycle. Odd parity is provided on the address extension bus by +ADREXTP.

During a data cycle, +ADREXT0 and +ADREXT1 are used as control inputs to ROMP Systems which implement 32-bit addressing will drive +ADREXT0 inactive during data cycles, indicating that 32-bit addressing is being used. These systems must also provide parity checking of the address extension bus for each address cycle. If a parity error is detected on the address extension bus for a given address cycle, +ADREXT1 is driven inactive during the following data cycle to indicate a parity error. If no parity error is detected, +ADREXT1 is driven active during the data cycle to indicate good parity. See "Address Extension Bus Definition" on page 99 for more information.

12.1.1.4 Exception

The state of +EXCEPTION during a reply, indicates whether the requested operation was valid. If +EXCEPTION is active during a reply, the request which produced the reply was invalid. If +EXCEPTION is inactive during the reply, the requested operation completed successfully. If +EXCEPTION is active during a reply to ROMP , a program check interrupt will result. For more information on +EXCEPTION see "Storage Protection and Address Translation" on page 109.

12.1.1.5 RSC Acknowledge and Not Acknowledge

Four control lines are used to indicate the results of transfers from one device to another on the RSC. Two lines (-ACKA and -NAKA) are used to indicate the results of address cycle transfers, and two other lines (-ACKD and -NAKD) are used to indicate the results of data cycle transfers. The four combinations of -ACKA/-ACKD and -NAKA/-NAKD are defined below:

ACKA/ ACKD	NAKA/ NAKD	Meaning
Inactive	Inactive	No Device Responded
Inactive	Active	Device Busy, Retry Transfer
Active	Inactive	Transfer Successful
Active	Active	Parity Error

See "Storage Channel I/O Pin Summary" on page 110 for more information on the acknowledge (-ACKA/-ACKD) and not acknowledge (-NAKA/-NAKD) handshake lines.

12.1.1.6 RSC Arbitration

Address Grant and Data Grant are groups of signals that are used to arbitrate among the devices on the RSC for use of the bus. Address Grant is used to arbitrate for an address cycle and the next data cycle. Data Grant is used to arbitrate for a data cycle. Address Grant and Data Grant are serially connected between devices, starting with the highest priority device and ending with the lowest priority device so that only one device at a time may originate transfer on the bus. This means that the Address Grant input (ADDRGRTI) of a given device in the priority chain is connected to the Address Grant output (ADDRGRTO) of the next higher priority device. Similarly, the Data Grant input (DATAGRTI) of one device is connected to the Data Grant output (DATAGRTO) of the next higher priority device. The lowest order DATAGRTO output (other than ROMP) is sent to all devices in the

Address Grant chain, and serves to keep them from using a data cycle needed for a reply. See "Bus Arbitration" on page 102 for more information on RSC arbitration. Figure 35 shows a typical connection of devices on the RSC.

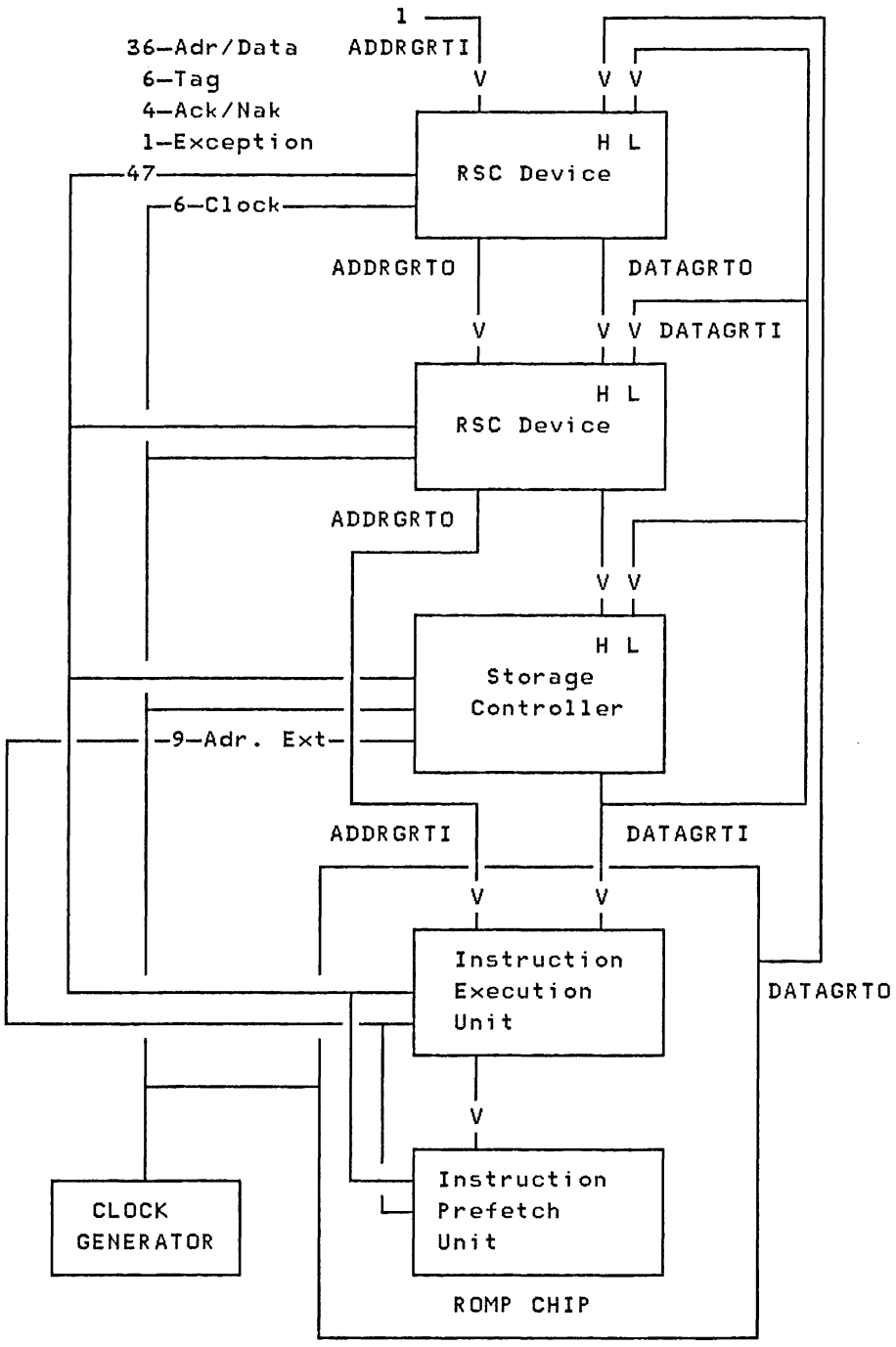


Figure 35. Typical RSC Configuration

12.1.1.7 Hold RSC

The -HOLD RSC input to ROMP is used to inhibit the timeout counter. The -HOLD RSC input is driven active by any device which can interfere with ROMP access to the RSC for an extended period of time. See "Hold Time-Out Counter" on page 108 for more information.

12.1.2 Clocks

ROMP requires a four phase clock (-T0, -T1, -T2, -T3), and two register file clocks (-W, -R), all of which are generated external to the processor chip. In addition, an ADDRESS CLOCK (+AC) and DATA CLOCK (+DC) are provided to enable the RSC tri-state drivers. ROMP , storage, and any other device on the RSC must use these clocks to control channel transfers. The timing relationships of these clocks are shown in Figure 36 on page 167. A detailed timing diagram for these clocks can be found in the ROMP Engineering Specification (ROMP E-SPEC). See "ROMP Engineering Specification" on page 188 for a reference to the ROMP E-SPEC.

12.1.3 Power

Power requirements for ROMP include a +5 volt supply, a +3.4 volt supply and a -3 volt supply (for substrate bias). Only a +5 volt supply is required to the card, all other voltages are derived from the +5 volt supply.

The +3.4 volt supply is derived from the +5 volt supply via an on-chip voltage regulator. The regulator requires an on card PNP darlington power transistor to supply the load current. The regulator functionally consists of a reference generator, an error amplifier and a driver. +3.4 VREF is connected to the collector of the power transistor and is compared to the output of the reference generator to obtain an error signal. This error signal (after biasing in the driver section) provides the regulation for the base of the power transistor and is represented in Figure 12.1 as +3.4 VREG. The emitter of the power transistor is connected to the +5 volt supply and the collector is used as the +3.4 volt supply.

The substrate bias supply is also derived from the 5-volt supply via an on-chip charge pump circuit, requiring no external components. The substrate voltage (-VSUB) is made available external to the chip.

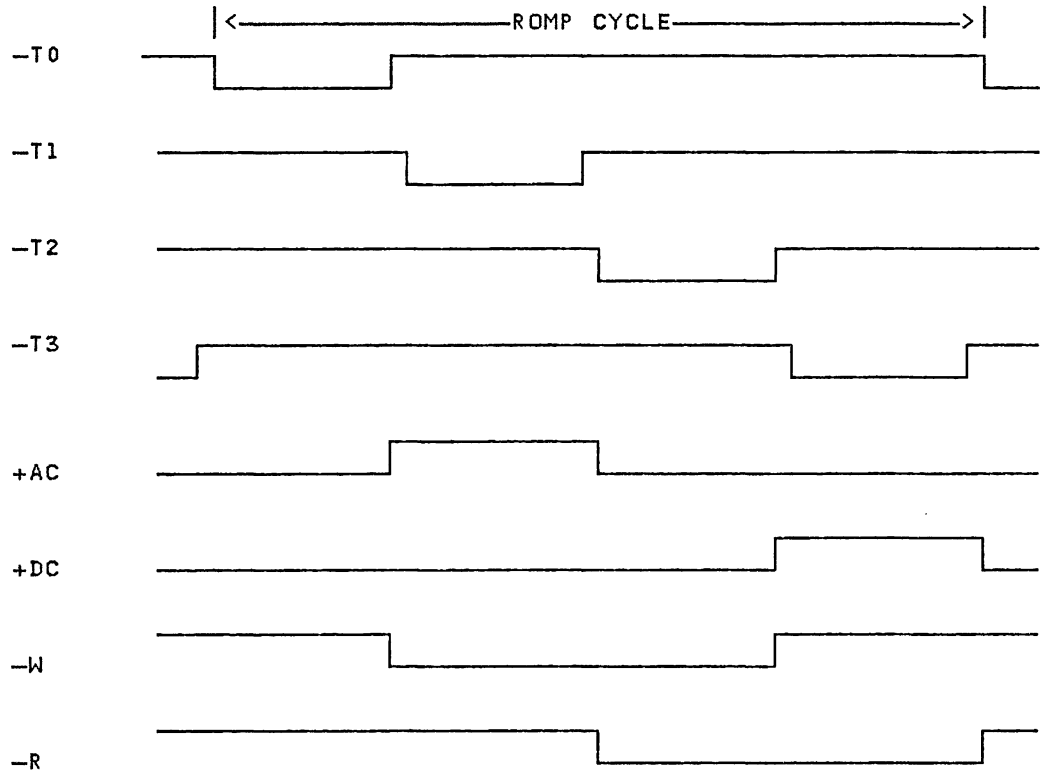


Figure 36. Clock Timing

12.1.4 Interrupt Inputs

The interrupt request inputs (-REQI 0-6) to ROMP permit the processor to change its status at the request of other system components. Each of the seven inputs defines a unique priority level, such that REQI0 is the highest priority interrupt, and REQI6 is the lowest priority interrupt. If more than one interrupt line is active, the lower priority interrupt is ignored until the higher priority interrupt has been serviced. For more information on interrupts and interrupt handling, see "Interrupts" on page 17.

The -TRAP input to ROMP is used by an I/O device to indicate a hardware error. Activating -TRAP causes a machine-check interrupt as described in "Machine-Check Error Handling" on page 122 and causes bit 22 of the the Machine Check Status (MCS) to be set to one by ROMP.

The interrupt request inputs (-REQI 0-6) are level sensitive, and should be driven active and held active by external devices until the interrupt request is serviced. The -TRAP input is edge sensitive, and should be driven active for one ROMP cycle. Devices activating the -TRAP input should deactivate -TRAP as soon as possible to allow reporting of subsequent machine check errors.

Note that activating the -TRAP input causes a machine check interrupt, and that a subsequent transition of the -TRAP input to an active state while on the machine check level will cause a check stop.

12.1.5 ROMP Controls

This section describes the various control signals used by the ROMP processor.

12.1.5.1 IPL Ready

The -IPL READY output is driven active by ROMP after successful completion of the internal microcode diagnostic. The -IPL READY output can be sensed by an IPL device (disk, diskette, etc.) to indicate that storage can be loaded by the device. -IPL READY is brought inactive one cycle after -IPL COMPLETE is activated. For more information see "Initial Program Load" on page 119.

12.1.5.2 IPL Complete

The -IPL COMPLETE input to ROMP is activated by the IPL device after loading of storage is complete and causes an IAR load to occur (see "IAR Load" on page 119) which then causes program execution to begin. -IPL READY is is brought inactive one cycle after -IPL COMPLETE is activated. For more information see "Initial Program Load" on page 119.

Systems which do not use an IPL device (i.e. those which contain IPL code in ROS) can tie -IPL COMPLETE active.

12.1.5.3 Fail

The -FAIL output indicates ROMP has entered the check stop state due to either a failure detected during the internal microcode diagnostic routine at IPL time, or due to a system error which was detected after IPL. The -FAIL output from ROMP is initialized to an active state by power-on reset. If an error is detected during the internal microcode diagnostic routine, ROMP enters the check stop state (see "Executing, Wait, Check Stop, and Stopped State" on page 7) and the -FAIL pin remains active. If no errors are

detected during the microcode diagnostic, the -FAIL pin is driven inactive. If ROMP enters the check stop state due to a system error at any time after IPL, the -FAIL output will go active. The -FAIL output can be sensed by an external device to detect processor failure.

12.1.5.4 Instruction Complete

The -INST CMPLT output from ROMP indicates that an instruction has completed execution. A transition from inactive to active occurs when execution of an instruction is completed. The frequency of this signal represents instruction execution rate (i.e. 1Mhz represents 1 MIP, 2.5Mhz represents 2.5 MIPs).

12.1.5.5 Sync

The -SYNC output from ROMP is used as a trigger source by a support processor when using the sync function. See "Processor Support Functions" on page 142 for more information on support processor functions supported by ROMP

12.1.5.6 Stop

The -STOP output from ROMP is driven active when ROMP is in the stopped state (see "Executing, Wait, Check Stop, and Stopped State" on page 7). The ROMP clock generator stops the ROMP clocks when -STOP is active.

12.1.5.7 Timer Clock

The -TIMER CLOCK input to ROMP is connected to an external clock and is used to decrement an internal 32-bit counter (see "System Timer Facility" on page 14). Note that since microcode is used to update the timer, the timer clock frequency should be much slower than the CPU clock rate. A reasonable -TIMER CLOCK frequency is 1 Khz.

12.1.5.8 Wait

The -WAIT output from ROMP is driven active when ROMP is in the wait state. ROMP enters the wait state by executing a WAIT instruction. ROMP may be removed from the wait state only through the occurrence of an interrupt, error, or power-on reset.

12.1.5.9 Chip In Place

The -CHIP IN PLACE input is used for second level testing to disable all ROMP off chip drivers (OCDs). When -CHIP IN PLACE is active, all OCDs are placed in a high impedance state.

12.1.5.10 Scan Gate

The +SCAN GATE input is used for Level Sensitive Scan Design (LSSD) testing. When +SCAN GATE is active, data is clocked into the internal ROMP registers from the scan inputs. Data from the internal scan strings is available at the scan outputs. When ROMP is reset, the +SCAN GATE input is activated and used to scan a known state into internal ROMP registers. Note that during reset, +SCAN GATE must be held active long enough to guarantee that all internal registers have been initialized. The minimum time that +SCAN GATE must be active to reset ROMP is defined in "ROMP Engineering Specification" on page 188.

12.1.6 Scan Inputs and Scan Outputs

The scan inputs are used to scan data into the the five LSSD scan strings in ROMP . These inputs are used for ROMP module testing and for access of internal ROMP registers by a support processor. The five scan string inputs are -SCANI0-4. The five scan outputs (+SCANO0-4) are used to scan data out of the internal ROMP registers. The five scan inputs (-SCANI0-4) and five scan outputs (+SCANO0-4) are multiplexed with system signals. The selection between normal system signals and scan inputs and scan outputs is determined by the state of +SCAN GATE. When +SCAN GATE is inactive, the multiplexed pins function as normal system signals. When +SCAN GATE is active, these pins function as scan inputs and scan outputs. A detailed definition of the scan strings is contained in "ROMP Scan String Definition" on page 188.

12.2 ROMP CHIP PIN ASSIGNMENT

Figure 37 shows the ROMP module footprint. Pin assignment for the ROMP chip is given in "Processor Signal Description" on page 172.

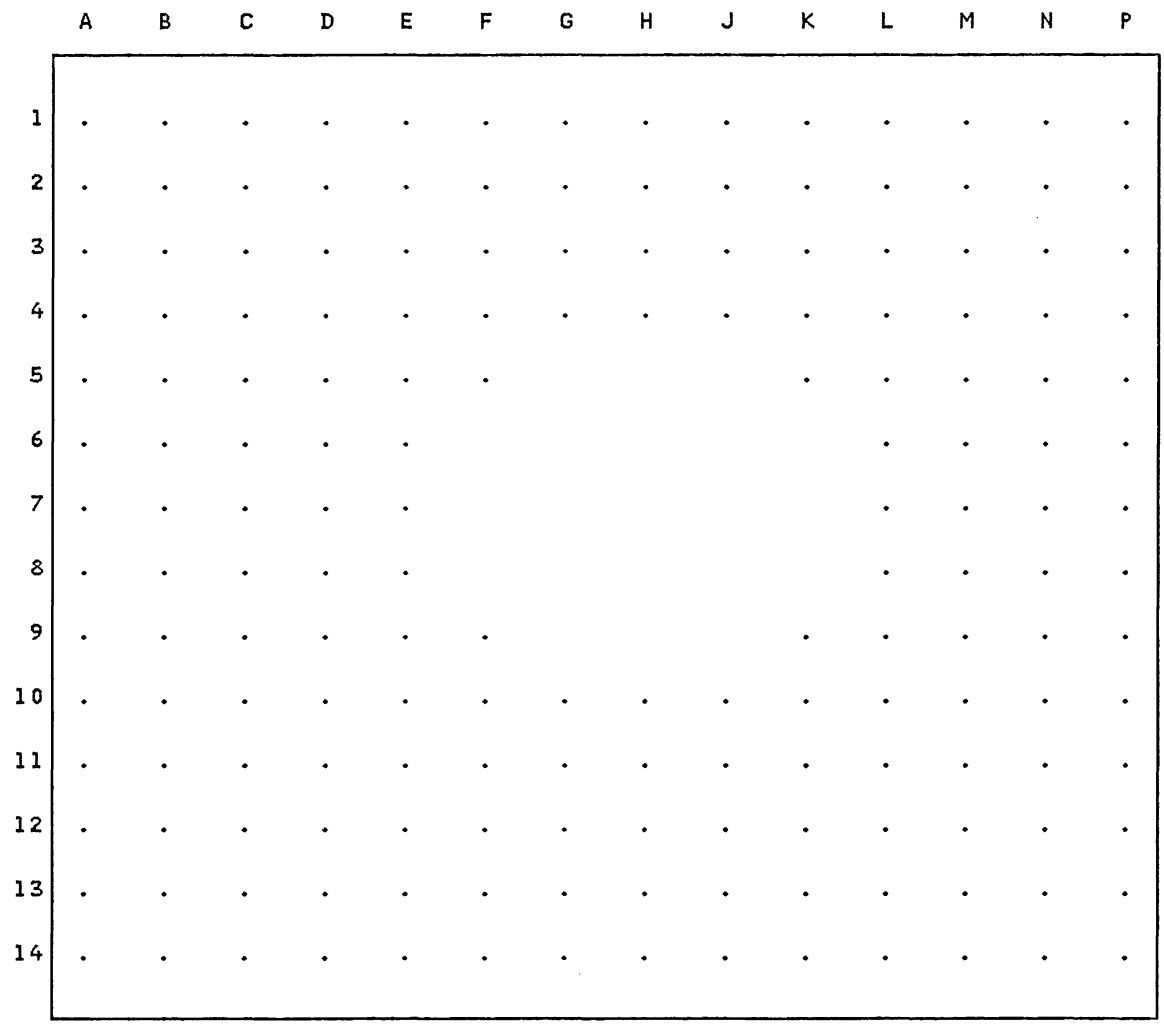


Figure 37. ROMP Module Footprint (Bottom View)

12.3 PROCESSOR SIGNAL DESCRIPTION

Signal Name	Direction	Driver Type	Pin Number
ROMP Storage Channel:			
+DAL00	B	TS	C10
+DAL01	B	TS	A12
+DAL02	B	TS	D09
+DAL03	B	TS	B13
+DAL04	B	TS	C06
+DAL05	B	TS	A08
+DAL06	B	TS	B08
+DAL07	B	TS	C08
+DAL08	B	TS	B06
+DAL09	B	TS	A06
+DAL10	B	TS	D08
+DAL11	B	TS	B11
+DAL12	B	TS	F11
+DAL13	B	TS	G12
+DAL14	B	TS	L14
+DAL15	B	TS	J12
+DAL16	B	TS	A05
+DAL17	B	TS	E06
+DAL18	B	TS	E08
+DAL19	B	TS	A10
+DAL20	B	TS	F12
+DAL21	B	TS	G11
+DAL22	B	TS	G13
+DAL23	B	TS	J14
+DAL24	B	TS	E13
+DAL25	B	TS	E14
+DAL26	B	TS	G14
+DAL27	B	TS	J13
+DAL28	B	TS	D11
+DAL29	B	TS	C12
+DAL30	B	TS	F10
+DAL31	B	TS	E12
+DALP0	B	TS	B14
+DALP1	B	TS	C14
+DALP2	B	TS	D13
+DALP3	B	TS	D14
+TAG0	B	TS	L04
+TAG1	B	TS	N02
+TAG2	B	TS	L03
+TAG3	B	TS	L02
+TAG4	B	TS	H01
+TAGP	B	TS	M04
-ACKA	B	TS	F01
-ACKD	B	TS	G02
-NAKA	B	TS	G03
-NAKD	B	TS	P03
ADDRGRTI	I	--	M13
DATAGRTI	I	--	J11

+DATAGRTO	O	AP	F02
-HOLD RSC	I	--	P05
+EXCEPTION	I	--	J02
+AC	I	--	L12
+DC	I	--	K13

ROMP Storage Channel Address Extension:

+ADREXT0	B	TS	A04
+ADREXT1	B	TS	B04
+ADREXT2	O	TS	D05
+ADREXT3	O	TS	C04
+ADREXT4	O	TS	A02
+ADREXT5	O	TS	C03
+ADREXT6	O	TS	D04
+ADREXT7	O	TS	B02
+ADREXTP	O	TS	D03

Clock Inputs:

-T0	I	--	N14
-T1	I	--	P10
-T2	I	--	P09
-T3	I	--	L10
-T3RF	I	--	P13
-W	I	--	P11
-R	I	--	N12

Control Signals:

-IPL READY	O	TS	D02
-IPL COMPLETE	I	--	P06
-FAIL	O	TS	C01
-INST CMPLT	O	TS	F03
-SYNC	O	TS	F04
-STOP	O	TS	D01
-TIMER CLOCK	I	--	P04
-WAIT	O	TS	G01

External Interrupt Inputs:

-REQI0	I	--	N03
-REQI1	I	--	M03
-REQI2	I	--	K04
-REQI3	I	--	M02
-REQI4	I	--	L01
-REQI5	I	--	K02
-REQI6	I	--	K01
-TRAP	I	--	M14

Scan Path and Controls:
(See note 1)

-SCANI0	I	--	N03
-SCANI1	I	--	M03
-SCANI2	I	--	K04
-SCANI3	I	--	M02
-SCANI4	I	--	L01
+SCANO0	O	TS	C01

+SCAN01	O	TS	F04
+SCAN02	O	TS	F03
+SCAN03	O	TS	F02
+SCAN04	O	TS	D01
+SCAN GATE	I	--	L11
-CHIP IN PLACE	I	--	P08

Power Connections:

+3.4VCKT1	D06
+3.4VCKT2	L09
+3.4VCKT3	M05
+3.4VOCD1	E05
+3.4VOCD2	E07
+3.4VOCD3	F09
+3.4VOCD4	G10
+3.4VOCD5	H04
+3.4VOCD6	L07
+5VCKT1	J10
+5VCKT2	K05
+5VCKT3	L08
+5VCKT4	M06
+5VCKT5	J04
+5VOCD1	E09
+5VOCD2	F05
GNDCKT1	K09
GNDCKT2	K10
GNDCKT3	L05
GNDCKT4	M08
GNDCKT5	J03
GNDOCD1	D07
GNDOCD2	E04
GNDOCD3	E10
GNDOCD4	G04
GNDOCD5	H10
GNDOCD6	L06
-3VSUB	H03
+3.4VREG	K03
+3.4VREF	H02

Notes:
- B = Bidirectional
- I = Input
- O = Output
- TS = Tristate
- AP = Active Pullup

1. Previously defined pins are used for scan-in and scan-out while the processor is in scan mode. These pins function only as scan-in and scan-out while SCAN GATE is active. Use of these pins is summarized below.

Pin	System Function	Scan Function
N03	-REQI0	-SCANI0
M03	-REQI1	-SCANI1
K04	-REQI2	-SCANI2
M02	-REQI3	-SCANI3
L01	-REQI4	-SCANI4
C01	-FAIL	+SCANO0
F04	-SYNC	+SCANO1
F03	-INST CMPLT	+SCANO2
F02	+DATAGRTO	+SCANO3
D01	-STOP	+SCANO4

13.0 APPENDIX

13.1 INSTRUCTION INDEX BY MNEMONIC

MNEMONIC	OP-CODE	FORMAT	PAGE	INSTRUCTION
A	E1	(R)	57	ADD
ABS	E0	(R)	59	ABSOLUTE
AE	F1	(R)	57	ADD EXTENDED
AEI	D1	(D)	58	ADD EXTENDED IMMEDIATE
AI	C1	(D)	58	ADD IMMEDIATE
AIS	90	(R)	58	ADD IMMEDIATE SHORT
BALA	8A	(BA)	42	BRANCH AND LINK ABSOLUTE
BALAX	8B	(BA)	42	BRANCH AND LINK ABSOLUTE WITH EXECUTE
BALI	8C	(BI)	42	BRANCH AND LINK IMMEDIATE
BALIX	8D	(BI)	43	BRANCH AND LINK IMMEDIATE WITH EXECUTE
BALR	EC	(R)	43	BRANCH AND LINK
BALRX	ED	(R)	44	BRANCH AND LINK WITH EXECUTE
BB	8E	(BI)	45	BRANCH ON CONDITION BIT IMMEDIATE
BBR	EE	(R)	46	BRANCH ON CONDITION BIT
BBRX	EF	(R)	46	BRANCH ON CONDITION BIT WITH EXECUTE
BBX	8F	(BI)	45	BRANCH ON CONDITION BIT IMMEDIATE WITH EXECUTE
BNB	88	(BI)	47	BRANCH ON NOT CONDITION BIT IMMEDIATE
BNBR	E8	(R)	48	BRANCH ON NOT CONDITION BIT
BNBRX	E9	(R)	48	BRANCH ON NOT CONDITION BIT WITH EXECUTE
BNBX	89	(BI)	47	BRANCH ON NOT CONDITION BIT IMMEDIATE WITH EXECUTE
C	B4	(R)	60	COMPARE
CAL	C8	(D)	37	COMPUTE ADDRESS LOWER HALF
CAL16	C2	(D)	37	COMPUTE ADDRESS LOWER HALF 16-BIT
CAS	6	(X)	38	COMPUTE ADDRESS SHORT
CAU	D8	(D)	38	COMPUTE ADDRESS UPPER HALF
CA16	F3	(R)	38	COMPUTE ADDRESS 16-BIT
CI	D4	(D)	61	COMPARE IMMEDIATE
CIS	94	(R)	61	COMPARE IMMEDIATE SHORT
CL	B3	(R)	62	COMPARE LOGICAL
CLI	D3	(D)	62	COMPARE LOGICAL IMMEDIATE
CLRBL	99	(R)	69	CLEAR BIT LOWER HALF
CLRBU	98	(R)	69	CLEAR BIT UPPER HALF
CLRSB	95	(R)	84	CLEAR SCR BIT
CLZ	F5	(R)	75	COUNT LEADING ZEROS
D	B6	(R)	65	DIVIDE STEP
DEC	93	(R)	39	DECREMENT
EXTS	B1	(R)	63	EXTEND SIGN
INC	91	(R)	39	INCREMENT
IOR	CB	(D)	88	INPUT/OUTPUT READ
IOW	DB	(D)	89	INPUT/OUTPUT WRITE
JB	08-0F	(JI)	44	JUMP ON CONDITION BIT

JNB	00-07	(JI)	46	JUMP ON NOT CONDITION BIT
L	CD	(D)	33	LOAD
LC	CE	(D)	31	LOAD CHARACTER
LCS	4	(DS)	31	LOAD CHARACTER SHORT
LH	DA	(D)	32	LOAD HALF
LHA	CA	(D)	32	LOAD HALF ALGEBRAIC
LHAS	5	(DS)	31	LOAD HALF ALGEBRAIC SHORT
LHS	EB	(R)	32	LOAD HALF SHORT
LIS	A4	(R)	39	LOAD IMMEDIATE SHORT
LM	C9	(D)	33	LOAD MULTIPLE
LPS	D0	(D)	85	LOAD PROGRAM STATUS
LS	7	(DS)	33	LOAD SHORT
M	E6	(R)	67	MULTIPLY STEP
MC03	F9	(R)	51	MOVE CHARACTER ZERO FROM THREE
MC13	FA	(R)	51	MOVE CHARACTER ONE FROM THREE
MC23	FB	(R)	52	MOVE CHARACTER TWO FROM THREE
MC33	FC	(R)	52	MOVE CHARACTER THREE FROM THREE
MC30	FD	(R)	52	MOVE CHARACTER THREE FROM ZERO
MC31	FE	(R)	53	MOVE CHARACTER THREE FROM ONE
MC32	FF	(R)	53	MOVE CHARACTER THREE FROM TWO
MFS	96	(R)	84	MOVE FROM SCR
MFTB	BC	(R)	53	MOVE FROM TEST BIT
MFTBIL	9D	(R)	54	MOVE FROM TEST BIT IMMEDIATE LOWER HALF
MFTBIU	9C	(R)	54	MOVE FROM TEST BIT IMMEDIATE UPPER HALF
MTS	B5	(R)	83	MOVE TO SCR
MTTB	BF	(R)	54	MOVE TO TEST BIT
MTTBIL	9F	(R)	55	MOVE TO TEST BIT IMMEDIATE LOWER HALF
MTTBIU	9E	(R)	55	MOVE TO TEST BIT IMMEDIATE UPPER HALF
N	E5	(R)	71	AND
NILO	C6	(D)	71	AND IMMEDIATE LOWER HALF EXTENDED ONES
NILZ	C5	(D)	71	AND IMMEDIATE LOWER HALF EXTENDED ZEROES
NIUO	D6	(D)	72	AND IMMEDIATE UPPER HALF EXTENDED ONES
NIUZ	D5	(D)	72	AND IMMEDIATE UPPER HALF EXTENDED ZEROES
O	E3	(R)	72	OR
OIL	C4	(D)	73	OR IMMEDIATE LOWER HALF
OIU	C3	(D)	73	OR IMMEDIATE UPPER HALF
ONEC	F4	(R)	59	ONE'S COMPLEMENT
S	E2	(R)	63	SUBTRACT
SAR	B0	(R)	76	SHIFT ALGEBRAIC RIGHT
SARI	A0	(R)	77	SHIFT ALGEBRAIC RIGHT IMMEDIATE
SARI16	A1	(R)	77	SHIFT ALGEBRAIC RIGHT IMMEDIATE PLUS SIXTEEN
SE	F2	(R)	64	SUBTRACT EXTENDED
SETBL	9B	(R)	70	SET BIT LOWER HALF
SETBU	9A	(R)	70	SET BIT UPPER HALF
SETSB	97	(R)	85	SET SCR BIT

SF	B2	(R)	63	SUBTRACT FROM
SFI	D2	(D)	64	SUBTRACT FROM IMMEDIATE
SIS	92	(R)	65	SUBTRACT IMMEDIATE SHORT
SL	BA	(R)	80	SHIFT LEFT
SLI	AA	(R)	80	SHIFT LEFT IMMEDIATE
SLI16	AB	(R)	81	SHIFT LEFT IMMEDIATE PLUS SIXTEEN
SLP	BB	(R)	81	SHIFT LEFT PAIRED
SLPI	AE	(R)	81	SHIFT LEFT PAIRED IMMEDIATE
SLPI16	AF	(R)	82	SHIFT LEFT PAIRED IMMEDIATE PLUS SIXTEEN
SR	B8	(R)	77	SHIFT RIGHT
SRI	A8	(R)	78	SHIFT RIGHT IMMEDIATE
SRI16	A9	(R)	78	SHIFT RIGHT IMMEDIATE PLUS SIXTEEN
SRP	B9	(R)	79	SHIFT RIGHT PAIRED
SRPI	AC	(R)	79	SHIFT RIGHT PAIRED IMMEDIATE
SRPI16	AD	(R)	79	SHIFT RIGHT PAIRED IMMEDIATE PLUS SIXTEEN
ST	DD	(D)	36	STORE
STC	DE	(D)	35	STORE CHARACTER
STCS	1	(DS)	34	STORE CHARACTER SHORT
STH	DC	(D)	35	STORE HALF
STHS	2	(DS)	35	STORE HALF SHORT
STM	D9	(D)	36	STORE MULTIPLE
STS	3	(DS)	36	STORE SHORT
SVC	C0	(D)	87	SUPERVISOR CALL
TGTE	BD	(R)	TGTE	TRAP IF REGISTER GREATER THAN OR EQUAL
TI	CC	(D)	49	TRAP ON CONDITION IMMEDIATE
TLT	BE	(R)	50	TRAP IF REGISTER LESS THAN
TSH	CF	(D)	34	TEST AND SET HALF
TWOC	E4	(R)	60	TWOS COMPLEMENT
WAIT	F0	(R)	86	WAIT
X	E7	(R)	73	EXCLUSIVE OR
XIL	C7	(D)	74	EXCLUSIVE OR IMMEDIATE LOWER HALF
XIU	D7	(D)	74	EXCLUSIVE OR IMMEDIATE UPPER HALF

13.2 INSTRUCTION INDEX BY OP CODE

OP-CODE	MNEMONIC	FORMAT	PAGE	INSTRUCTION
00-07	JNB	(JI)	46	JUMP ON NOT CONDITION BIT
08-0F	JB	(JI)	44	JUMP ON CONDITION BIT
1	STCS	(DS)	34	STORE CHARACTER SHORT
2	STHS	(DS)	35	STORE HALF SHORT
3	STS	(DS)	36	STORE SHORT
4	LCS	(DS)	31	LOAD CHARACTER SHORT
5	LHAS	(DS)	31	LOAD HALF ALGEBRAIC SHORT
6	CAS	(X)	38	COMPUTE ADDRESS SHORT
7	LS	(DS)	33	LOAD SHORT
80-87	RESERVED			
88	BNB	(BI)	47	BRANCH ON NOT CONDITION BIT IMMEDIATE
89	BNBX	(BI)	47	BRANCH ON NOT CONDITION BIT IMMEDIATE WITH EXECUTE
8A	BALA	(BA)	42	BRANCH AND LINK ABSOLUTE
8B	BALAX	(BA)	42	BRANCH AND LINK ABSOLUTE WITH EXECUTE
8C	BALI	(BI)	42	BRANCH AND LINK IMMEDIATE
8D	BALIX	(BI)	43	BRANCH AND LINK IMMEDIATE WITH EXECUTE
8E	BB	(BI)	45	BRANCH ON CONDITION BIT IMMEDIATE
8F	BBX	(BI)	45	BRANCH ON CONDITION BIT IMMEDIATE WITH EXECUTE
90	AIS	(R)	58	ADD IMMEDIATE SHORT
91	INC	(R)	39	INCREMENT
92	SIS	(R)	65	SUBTRACT IMMEDIATE SHORT
93	DEC	(R)	39	DECREMENT
94	CIS	(R)	61	COMPARE IMMEDIATE SHORT
95	CLRSB	(R)	84	CLEAR SCR BIT
96	MFS	(R)	84	MOVE FROM SCR
97	SETSB	(R)	85	SET SCR BIT
98	CLRBU	(R)	69	CLEAR BIT UPPER HALF
99	CLRBL	(R)	69	CLEAR BIT LOWER HALF
9A	SETBU	(R)	70	SET BIT UPPER HALF
9B	SETBL	(R)	70	SET BIT LOWER HALF
9C	MFTBIU	(R)	53	MOVE FROM TEST BIT IMMEDIATE UPPER HALF
9D	MFTBIL	(R)	54	MOVE FROM TEST BIT IMMEDIATE LOWER HALF
9E	MTTBIU	(R)	55	MOVE TO TEST BIT IMMEDIATE UPPER HALF
9F	MTTBIL	(R)	55	MOVE TO TEST BIT IMMEDIATE LOWER HALF
A0	SARI	(R)	77	SHIFT ALGEBRAIC RIGHT IMMEDIATE
A1	SARI16	(R)	77	SHIFT ALGEBRAIC RIGHT IMMEDIATE PLUS SIXTEEN
A2,A3	RESERVED			
A4	LIS	(R)	39	LOAD IMMEDIATE SHORT
A5-A7	RESERVED			

A8	SRI	(R)	78	SHIFT RIGHT IMMEDIATE
A9	SRI16	(R)	78	SHIFT RIGHT IMMEDIATE PLUS SIXTEEN
AA	SLI	(R)	80	SHIFT LEFT IMMEDIATE
AB	SLI16	(R)	81	SHIFT LEFT IMMEDIATE PLUS SIXTEEN
AC	SRPI	(R)	79	SHIFT RIGHT PAIRED IMMEDIATE
AD	SRPI16	(R)	79	SHIFT RIGHT PAIRED IMMEDIATE PLUS SIXTEEN
AE	SLPI	(R)	81	SHIFT LEFT PAIRED IMMEDIATE
AF	SLPI16	(R)	82	SHIFT LEFT PAIRED IMMEDIATE PLUS SIXTEEN
B0	SAR	(R)	76	SHIFT ALGEBRAIC RIGHT
B1	EXTS	(R)	63	EXTEND SIGN
B2	SF	(R)	63	SUBTRACT FROM
B3	CL	(R)	62	COMPARE LOGICAL
B4	C	(R)	60	COMPARE
B5	MTS	(R)	83	MOVE TO SCR
B6	D	(R)	65	DIVIDE STEP
B7	RESERVED			
B8	SR	(R)	77	SHIFT RIGHT
B9	SRP	(R)	79	SHIFT RIGHT PAIRED
BA	SL	(R)	80	SHIFT LEFT
BB	SLP	(R)	81	SHIFT LEFT PAIRED
BC	MFTB	(R)	53	MOVE FROM TEST BIT
BD	TGTE	(R)	50	TRAP IF GREATER THAN OR EQUAL
BE	TLT	(R)	50	TRAP IF LESS THAN
BF	MTTB	(R)	54	MOVE TO TEST BIT
C0	SVC	(D)	87	SUPERVISOR CALL
C1	AI	(D)	58	ADD IMMEDIATE
C2	CAL16	(D)	37	COMPUTE ADDRESS LOWER HALF 16-BIT
C3	OIU	(D)	73	OR IMMEDIATE UPPER HALF
C4	OIL	(D)	73	OR IMMEDIATE LOWER HALF
C5	NILZ	(D)	71	AND IMMEDIATE LOWER HALF EXTENDED ZEROES
C6	NILO	(D)	71	AND IMMEDIATE LOWER HALF EXTENDED ONES
C7	XIL	(D)	74	EXCLUSIVE OR IMMEDIATE LOWER HALF
C8	CAL	(D)	37	COMPUTE ADDRESS LOWER HALF
C9	LM	(D)	33	LOAD MULTIPLE
CA	LHA	(D)	32	LOAD HALF ALGEBRAIC
CB	IOR	(D)	88	INPUT/OUTPUT READ
CC	TI	(D)	49	TRAP ON CONDITION IMMEDIATE
CD	L	(D)	33	LOAD
CE	LC	(D)	31	LOAD CHARACTER
CF	TSH	(D)	34	TEST AND SET HALF
D0	LPS	(D)	85	LOAD PROGRAM STATUS
D1	AEI	(D)	58	ADD EXTENDED IMMEDIATE
D2	SFI	(D)	64	SUBTRACT FROM IMMEDIATE
D3	CLI	(D)	62	COMPARE LOGICAL IMMEDIATE
D4	CI	(D)	61	COMPARE IMMEDIATE
D5	NIUZ	(D)	72	AND IMMEDIATE UPPER HALF EXTENDED ZEROES
D6	NIUO	(D)	72	AND IMMEDIATE UPPER HALF EXTENDED ONES
D7	XIU	(D)	74	EXCLUSIVE OR IMMEDIATE UPPER HALF

D8	CAU	(D)	38	COMPUTE ADDRESS UPPER HALF
D9	STM	(D)	36	STORE MULTIPLE
DA	LH	(D)	32	LOAD HALF
DB	IOW	(D)	89	INPUT/OUTPUT WRITE
DC	STH	(D)	35	STORE HALF
DD	ST	(D)	36	STORE
DE	STC	(D)	35	STORE CHARACTER
DF	RESERVED			
E0	ABS	(R)	59	ABSOLUTE
E1	A	(R)	57	ADD
E2	S	(R)	63	SUBTRACT
E3	O	(R)	72	OR
E4	TWOC	(R)	60	TWOS COMPLEMENT
E5	N	(R)	71	AND
E6	M	(R)	67	MULTIPLY STEP
E7	X	(R)	73	EXCLUSIVE OR
E8	BNBR	(R)	48	BRANCH ON NOT CONDITION BIT
E9	BNBRX	(R)	48	BRANCH ON NOT CONDITION BIT WITH EXECUTE
EA	RESERVED			
EB	LHS	(R)	32	LOAD HALF SHORT
EC	BALR	(R)	43	BRANCH AND LINK
ED	BALRX	(R)	44	BRANCH AND LINK WITH EXECUTE
EE	BBR	(R)	46	BRANCH ON CONDITION BIT
EF	BBRX	(R)	46	BRANCH ON CONDITION BIT WITH EXECUTE
F0	WAIT	(R)	86	WAIT
F1	AE	(R)	57	ADD EXTENDED
F2	SE	(R)	64	SUBTRACT EXTENDED
F3	CA16	(R)	38	COMPUTE ADDRESS 16-BIT
F4	ONEC	(R)	59	ONES COMPLEMENT
F5	CLZ	(R)	75	COUNT LEADING ZEROS
F6-F8	RESERVED			
F9	MC03	(R)	51	MOVE CHARACTER ZERO FROM THREE
FA	MC13	(R)	51	MOVE CHARACTER ONE FROM THREE
FB	MC23	(R)	52	MOVE CHARACTER TWO FROM THREE
FC	MC33	(R)	52	MOVE CHARACTER THREE FROM THREE
FD	MC30	(R)	52	MOVE CHARACTER THREE FROM ZERO
FE	MC31	(R)	53	MOVE CHARACTER THREE FROM ONE
FF	MC32	(R)	53	MOVE CHARACTER THREE FROM TWO

13.3 PRIVILEGED INSTRUCTIONS

MNEMONIC	OP-CODE	FORMAT	PAGE	INSTRUCTION	NOTE
CLRSB	95	(R)	84	CLEAR SCR BIT	1
LPS	D0	(D)	85	LOAD PROGRAM STATUS	
MFS	96	(R)	84	MOVE FROM SCR	1
MTS	B5	(R)	83	MOVE TO SCR	1
SETSB	97	(R)	85	SET SCR BIT	1
WAIT	F0	(R)	86	WAIT	

Notes: 1. Clear SCR Bit (CLRSB), Move From SCR (MFS), Move To SCR (MTS), and Set SCR Bit (SETSB) are privileged if the referenced SCR is the Counter Source (SCR 6), Counter (SCR7), Timer Status (SCR8), Machine Check Status (SCR 11), Program Check Status (SCR 11), Interrupt Request Buffer (SCR 12), Instruction Address Register (SCR 13), or the Interrupt Control Status (SCR 14). Clear SCR Bit (CLRSB), Move From SCR (MFS), Move To SCR (MTS), and Set SCR Bit (SETSB) are non-privileged if the referenced SCR is the Multiplier Quotient (SCR 10), or the Condition Status (SCR 15).

13.4 ILLEGAL BRANCH WITH EXECUTE SUBJECT INSTRUCTIONS

MNEMONIC	OP-CODE	FORMAT	PAGE	INSTRUCTION
BALA	8A	(BA)	42	BRANCH AND LINK ABSOLUTE
BALAX	8B	(BA)	42	BRANCH AND LINK ABSOLUTE WITH EXECUTE
BALI	8C	(BI)	42	BRANCH AND LINK IMMEDIATE
BALIX	8D	(BI)	43	BRANCH AND LINK IMMEDIATE WITH EXECUTE
BALR	EC	(R)	43	BRANCH AND LINK
BALRX	ED	(R)	44	BRANCH AND LINK WITH EXECUTE
BB	8E	(BI)	45	BRANCH ON CONDITION BIT IMMEDIATE
BBR	EE	(R)	46	BRANCH ON CONDITION BIT
BBRX	EF	(R)	46	BRANCH ON CONDITION BIT WITH EXECUTE
BBX	8F	(BI)	45	BRANCH ON CONDITION BIT IMMEDIATE WITH EXECUTE
BNB	88	(BI)	47	BRANCH ON NOT CONDITION BIT IMMEDIATE
BNBR	E8	(R)	48	BRANCH ON NOT CONDITION BIT
BNBRX	E9	(R)	48	BRANCH ON NOT CONDITION BIT WITH EXECUTE
BNBX	89	(BI)	47	BRANCH ON NOT CONDITION BIT IMMEDIATE WITH EXECUTE
JB	08-0F	(JI)	44	JUMP ON CONDITION BIT
JNB	00-07	(JI)	46	JUMP ON NOT CONDITION BIT
LPS	D0	(D)	85	LOAD PROGRAM STATUS
SVC	C0	(D)	87	SUPERVISOR CALL
TGTE	BD	(R)	50	TRAP IF REGISTER GREATER THAN OR EQUAL
TI	CC	(D)	49	TRAP ON CONDITION IMMEDIATE
TLT	BE	(R)	50	TRAP IF REGISTER LESS THAN
WAIT	F0	(R)	86	WAIT

13.5 ROMP SYSTEM SUPPORT SOFTWARE

This section gives a brief description of the system support software for ROMP and the documentation available. Documentation described in this section can be obtained from Susan Strachan, Dept. 540, Bldg. 045, Austin, TX.

13.5.1 PL.8 Compiler

The PL.8 compiler is an optimizing compiler for a PL/I variant--a full high-level language designed to be suitable for both general applications and systems programming. Many inherently inefficient constructions of the PL/I language have been eliminated or modified so that the language can be compiled to efficient object code. The compiler incorporates state-of-the-art graph-flow analysis techniques which have not heretofore been implemented in a compiler. The compiler is very effective--it generates code which is only about 10% larger than well-tuned assembly code for sizeable modules. The compiler is at release-level reliability.

Documentation:

Online SCRIPT file PL8REF -- PL.8 Language Reference Manual
Online SCRIPT file PL8GUIDE -- PL.8 CMS User's Guide
Online SCRIPT file PL8LANG -- PL.8 Language Specifications
Online SCRIPT file PL8BNF -- PL.8 BNF Syntax Diagrams

13.5.2 PASCAL Compiler

The PASCAL compiler offers an alternative high-level language for the application programmer. The language's flexible data structures and well structured program control make PASCAL a very powerful tool suitable for most applications. The language was implemented according to PASCAL: USER MANUAL AND REPORT by Niklaus Wirth and Kathleen Jensen (Springer-Verlag: New York, 1974) with extensions to allow separate module compilation and linkage similar to that of PASCALVS.

The PASCAL compiler is the existing PL.8 compiler with a separate front end, thus it also produces efficient object code. This also allows free intermixing of PL.8 and PASCAL programs with shared support routines.

Documentation:

Online SCRIPT file PASCAL8 -- Pascal Language Reference Manual

13.5.3 C Compiler

The C compiler is in development and should be available 2Q84. The language is based on the C language described in "The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie.

13.5.4 ROMP Development System

Most of the ROMP software tools can run on ROMP/ROSETTA based hardware. The CPR operating system provides a multi-tasking, multiple virtual address space, programming environment with a VM/370 CMS-like file system and application environment. Support tools include the PL.8 compiler, the binder, and symbolic debugger. An editor, an EXEC interpreter, and file system utilities (erase, copy, rename, etc.) are also provided.

Documentation:

Online SCRIPT file RDSGUIDE -- ROMP Development System Users Guide

13.5.5 PL.8 Source Level Debugger

The source level debugger allows a programmer to debug PL.8 code at the source level. The debugger supports stop on a PL.8 statement, statement single step, variable inspection, and altering of variables. The debugger runs on CMS and CPR.

Documentation:

Online SCRIPT file PL8DEBUG -- PL.8 Source Level Debugger Documentation

13.5.6 PL.8 Machine-Level Program Analysis Tool

This program works in conjunction with the PL.8 Source Level Debugger and provides the following functions:

- Supplies low level machine dependent debugging mechanisms to the Source Level Debugger.
- Provides a consistent command environment for the end user debugging at a lower level.

Documentation:

Online SCRIPT file DEBUGM -- PL.8 Machine Level Program Analysis Tool

13.5.7 PL.8 Source And Design Code Formatter

This is a collection of programs to support system design and software development. The following functions are currently supported or under development:

1. Source code formatting.
2. Design code specification.
3. Design code extractor.
4. A STARCHART generator.
5. XEDIT macros that provide templates for PL.8 language constructs.

Online documentation will be available.

13.5.8 PL.8 Macro Pre-processor

The PL.8 Macro Pre-processor (PL.8MP) provides the compile-time macro capabilities of the PL/S III compiler macro pre-processor. One extension made allows INCLUDE statements within macro definitions. The macro pre-processor also supports a format option that re-formats a PL.8 file to a PL/S III form.

Documentation:

Online SCRIPT file PL8MP -- Differences between PL.8MP and PL/S III
Online SCRIPT file PL8MPLR -- PL.8MP Language Reference Manual

13.5.9 ROMP Assembler

The ROMP Assembler is a full macro assembler for ROMP based on the HSK Assembler. It is a cross-assembler which runs on VM/370.

Documentation:

Printed Manual - ROMP Assembler Language Manual

13.5.10 ROMP Simulator

The ROMP Simulator is a high performance simulator that runs on VM/370.

Documentation:

Online SCRIPT file RSIMINT -- Method of simulator operation.
CONSOLE RSIM --Self Documented EXEC that provides the user interface.

13.5.11 Program Binder For ROMP

The binder takes program text files as produced by the assemblers and compilers and binds them together into one or more programs. These output programs can then be loaded by a suitable relocatable loader. The Binder insures that parameters and arguments of external procedures have been declared the same.

Documentation:

Online SCRIPT file TOCBIND -- Description of the ROMP binder.

13.5.12 ROMP Hardware Development System

The Hardware Development System runs on the Series 1 or IBM PC and provides an interface to ROMP via the LSSD scan strings. This software provides the functions described in "Support Processor Facilities" on page 142. In addition to the software for the Series 1 or IBM PC, interface logic is provided for connection to the ROMP scan strings. Information on the Hardware Development System can be obtained from: Tom Whiteside, Dept. F61, Bldg. 045, Austin, TX.

13.5.13 Program Development Library (PDL) Interface

Interface EXECs which integrate the PL.8 compiler, ROMP assembler, and binder into the PDL library system are described in the online file: PDLPL8 SCRIPT.

13.5.14 RTIMER Simulator

The RTIMER Simulator provides performance analysis of specific ROMP system configurations, using specific instruction sequences. Input to the RTIMER Simulator consists of a definition of the system configuration (ROMP cycle time, storage organization, storage speed, etc.) and a user supplied trace tape of instructions to be executed. The trace tape is produced by the ROMP Simulator. Output from the RTIMER Simulator lists the overall performance and detailed analysis of operations during each simulation cycle.

13.6 ROMP SYSTEM HARDWARE REFERENCES

This section gives a brief description of the various hardware support documents available describing the ROMP hardware characteristics and interfaces.

13.6.1 ROMP Engineering Specification

The electrical and environmental characteristics are described in the ROMP ENGINEERING SPECIFICATION (ROMP E-SPEC). The ROMP E-SPEC can be obtained from: Pete Mc Cormick, Dept. N55 Bldg. 967-2, Burlington, VT.

13.6.2 ROMP Scan String Definition

The ROMP scan strings are defined in the ROMP Scan String Document, which can be obtained from: Mike Johnson, Dept. F60, Bldg 045, Austin, TX.

13.6.3 Support Processor Interface

The Support Processor Interface document defines a suggested standard interface between ROMP and a support processor, and can be obtained from: Tom Whiteside, Dept. F60, Bldg 045, Austin, TX.

13.6.4 ROMP AC Hardware Characterization Plan

This document describes the test strategy and detailed test plan leading to ROMP qualification (T2). In addition, this document contains various sections which describe the internal operation of ROMP. Copies of the ROMP AC Hardware Characterization Plan can be obtained from: Kanti Shah, Dept. F60, Bldg 045, Austin, TX.

End Of
Document