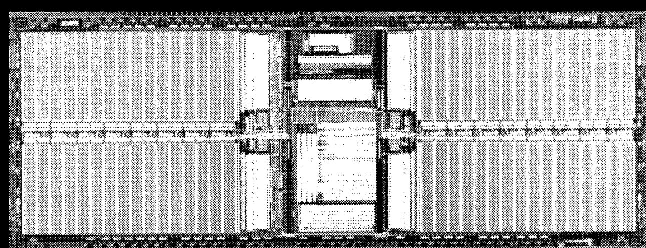
MITSUBISHI 32-bit RISC Microprocessor
with 2 Mbytes on-chip DRAM

M32R/D Family



USER'S MANUAL

MITSUBISHI
ELECTRONIC DEVICE GROUP

First Edition : January 12, 1996

Restricted Rights Legend

Mitsubishi Electric Corporation makes no warranty for the use of its products and assumes no responsibility for any errors that may appear in this manual nor does it make a commitment to update the information contained in this manual.

The information in this manual is subject to change without notice.

This manual is provided "as is" without warranty of any kind either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

Contents

Preface IX

1 Overview 1

2 Programming Model 19

3 Exception,Interrupt,Trap (EIT) 35

Preface

About This Manual

This manual describes the M32R family features, hardware functions, and software functions. For information about the development support tools for M32R, refer to the user's manual for the development support tools.

Organization of this Manual

This manual consists of :

Chapter 1, "Overview", describes the features (specifications and components) of the M32R family microprocessor.

Chapter 2, "Programming Model", describes the M32R application programming environment as seen by assembly language programmers.

Chapter 3, "Exception,Interrupt,Trap(EIT)", describes EIT Processing.

Chapter 4, "Internal Memory System", describes the internal memory system which consists of the DRAM, cache, and memory controller when each cache mode is selected.

Chapter 5, "Internal Peripheral I/O", describes I/O registers to control the internal memory system and programmable ports.

Chapter 6, "External Bus Interface", describes external bus interface signals, CPU bus cycles, and reset processing.

Chapter 7, "Electrical Characteristics", describes operating conditions of M32R family.

Appendix A, "The M32R Instruction Set", provides the M32R instruction set in alphabetical order.

Appendix B, "I/O Registers", provides configuration and functions of each I/O register.

Appendix C, "Pipeline Processing", describes the pipeline processing of M32R family.

Appendix D, "Instruction Execution Time", provides the number of cycles to perform instruction execution and memory access.

Conventions

The following conventions indicate operands of M32R's instructions ("x" means any notation. "*n*" means any number) :

Symbol	Meaning
R*n*	A general register (*n*=0~15).
CR*n*	A control register.
Rx+	Indicates to add contents of a general register and 4 (operand size) after to reference the general register. A "register indirect + register update" addressing mode.
+Rx	Indicates to add contents of a general register and 4 (operand size) before to reference the general register. A "register indirect + register update" addressing mode.
-Rx	Indicates to subtract 4 (operand size) from contents of a general register before to reference the general register. A "register indirect + register update" addressing mode.
Rsrc, Rsrc*n*	A referenced general register (has an address or a value etc. to process).
CRsrc	A referenced control register .
Rdst, CRdest	A destination register.
disp*n*	An *n*-bit displacement value.
imm*n*	An *n*-bit immediate data (signed integer).

The following conventions are, only in A.1 "List of Instructions", to indicate operands of M32R's instructions (" *n*" means any number) :

Symbol	Meaning
C*n*	An *n*-bit constant (e.g. displacement, immediate).
n	A number (e.g. trap number).

1

Overview

1.1 About M32R

1.1.1 Microprocessor the M32R with On-chip DRAM

The world's first microprocessor with on-chip DRAM

- The M32R is the first 32-bit microprocessor in the world with an on-chip 2M-byte large capacity DRAM. With a 32-bit RISC CPU as its core, the M32R has peripheral functions such as a memory controller in addition to a 2K-byte cache memory.
- With its I/O controller ASIC and program ROM chips, the M32R handles a wide range of applications such as data processing and device control.

High performance with on-chip 128-bit bus

- By including the DRAM, the CPU, DRAM and cache memory are connected with a 128-bit internal bus, thus making possible high speed transfer of instructions and data between the DRAM and the CPU.
- The CPU and the internal bus of the M32R operate at high frequency (up to 66.6MHz). As a result, a high performance of 52.4 VAX MIPS at a Dhrystone V2.1 bench mark is achieved when operating at 66.6MHz by using the on-chip DRAM.
- All items necessary to achieve a high-performance system are included in the M32R. High speed external memory and complex memory control required in conventional RISC microprocessors are not necessary. Performance including system cost can be suitable by using the M32R.

Low power consumption with 16-bit external bus

- M32R is provided with a 16-bit external data bus for ease in connecting external ROM and I/O controllers. The external address bus is 24 bits wide with an operating frequency of 16.6MHz (max.).
- Data traffic with an external I/O controller will be reduced in the M32R because of the on-chip DRAM. Furthermore, by using a 16-bit external bus and lowering its operating frequency, power to drive the external bus has been drastically reduced. Reducing system power consumption.
- Burst transfer mode with the bus is also supported with a 128-bit buffer in the external bus interface of the M32R. This provides high speed data transfer from an external ROM to the cache memory and from the on-chip DRAM to external circuits.

Operation with quadrupled input clock frequency

- The M32R uses an internal clock the input clock frequency. For example, if a 16.6MHz clock is input, the internal operating frequency will be 66.6MHz. By multiplying the clock frequency internally, low input clock frequencies can be used, thus facilitating system design.

1.1.2 High Performance, Compact RISC CPU

Employing RISC architecture

- RISC architecture is conformed to the CPU core of the M32R. Memory is accessed by using the load and store instructions and other operations are executed by using register-to-register operation instructions. A total of 83 instructions are available using sixteen 32-bit registers.
- The M32R supports compound instructions such as load and address update and store and address update which are useful for high speed data transfer.

5-stage pipeline processing

- The M32R executes instructions at high speed with 5-stage pipeline consisting of instruction fetch, decode, execute, memory access and write-back.
- Not only are load, store and operation instructions between registers executed but compound instructions such as load and address update and store and address update are also executed in one cycle.

- By using "out of order-completion" the M32R is capable of carrying out instruction execution control without wasting clock cycles. Instructions are put into the execution stage in the order fetched. However, when a wait cycle is inserted in the memory access of load instructions or store instruction, the following instruction between registers may be completed first.

Compact CPU

- The size of the CPU core excluding the multiply and accumulate unit is about 2mm X 2mm. By making the CPU compact, it was possible to build in the 2M byte large capacity DRAM.

Compact code

- The M32R has two types of instructions, 16 bit instructions and 32 bit instructions. By supporting the 16-bit instruction, it is possible to compress the code size and thus use space of the built-in DRAM more effectively.

- In the 32-bit instruction, the branch instruction used for direct branching from the address of the instruction being executed to the +/-32M byte linear address space is supported.

1.1.3 Multiply and Accumulate Unit for Digital Signal Processing (DSP)

High speed multiplier

- A 32-bit X 16-bit high-speed multiplier is built into the M32R, making it possible to execute integer multiply instructions of 32 bits X 32 bits in three cycles.

Multiply and accumulate instructions comparable to DSP

- The M32R has an on-chip 64-bit accumulator. The following four multiply and accumulate instruction types (also multiplying instructions) are possible with 56-bit data as the object and are each executed in one cycle.

  - High order 16 bits of register X high order 16 bits of register
  - Low order 16 bits of register X low order 16 bits of register
  - All 32 bits of register X high order 16 bits of register
  - All 32 bits of register X low order 16 bits of register

- The M32R supports instructions to round off 16 bit or 32 bit values stored in the accumulator which execute in 1 cycle.
- By combining these instructions with high speed data transfer instructions such as load and address update and store and address update, data processing will be possible for DSP applications.

1.1.4 Memory Controller Supporting On-chip and External Memories

Cache mode suitable for on-chip DRAM and external program ROM

- The M32R has a 2K-byte cache memory and memory controller which support the following two modes (Three modes including the non-cache mode) :
  - On-chip DRAM caching mode
    Unified cache of instruction/data of on-chip DRAM
  - External ROM caching mode
    Instruction cache relative to external program ROM
- The on-chip DRAM caching mode assumes a microprocessor system using the on-chip DRAM as its main memory. The 2K byte cache memory operates as a direct map system cache relative to the instruction/data stored in the on-chip DRAM. It operates as a store-in cache and holds data traffic between the cache memory and DRAM to a minimum.
- The external ROM caching mode assumes the use of an external ROM as the program memory and the on-chip DRAM as the data memory. The 2K byte memory operates as a direct map cache for instructions stored in the external program ROM. However, there is no caching of data.

DRAM control suitable for CPU operation mode

- By setting the operating frequency (input clock over or under 8.3 MHz), the memory controller optimizes access timing to the on-chip DRAM.
- The M32R also has an on-chip DRAM refresh control that supports the following two modes :
  - **Auto refresh mode**
    Refresh cycle is automatically inserted during operation of CPU (CAS before RAS)

- **Self-refresh mode**
  Starts self refresh cycle of DRAM during sleep of CPU to save on power consumption.

1.2 Block Diagram

The block diagram of the M32R is shown in Figure 1.1 (Section 1.3 lists features of each block).

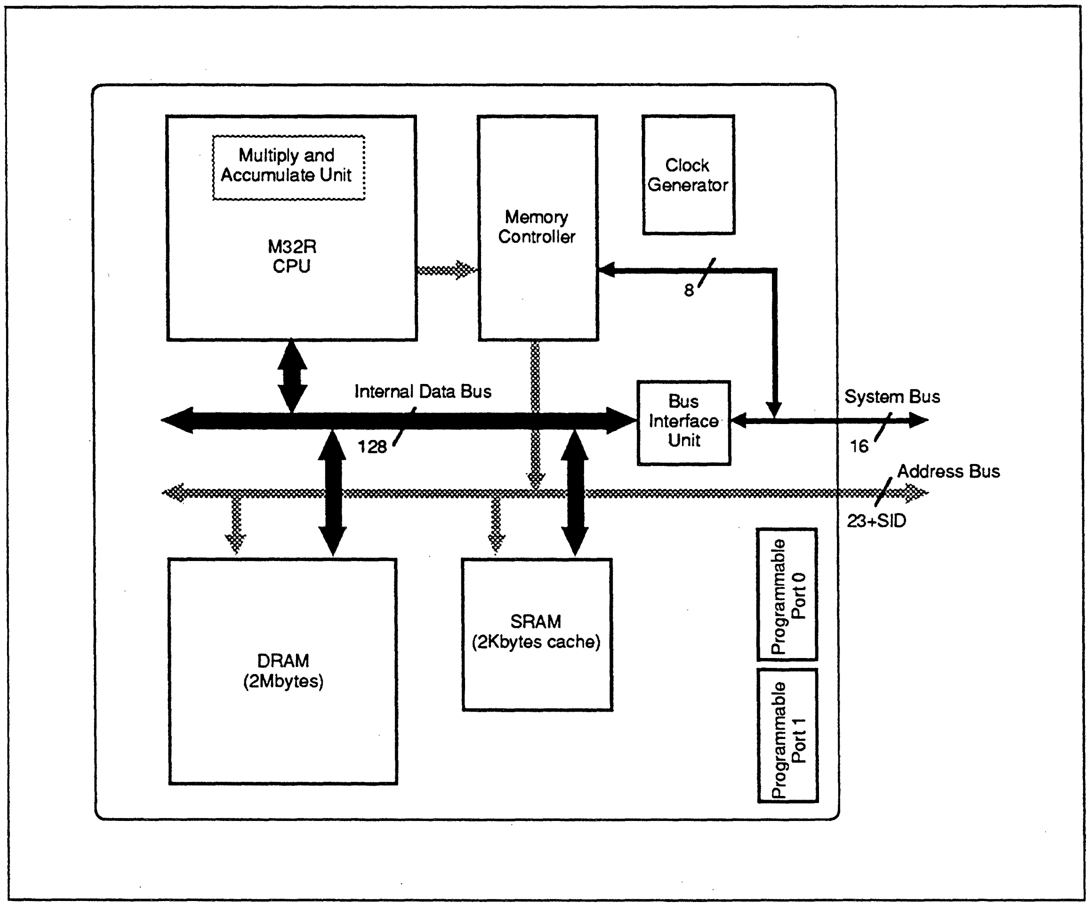


Figure 1.1 M32R Block Diagram

1.3 Device Specifications

The following specifications describe the devices in the M32R :

1.3.1 M32R CPU Core

Table 1.1 M32R Core Specifications

Item	Specification
Architecture	32-bit RISC architecture processor using 5-stage pipeline method (with the multiply and accumulate unit)
Bus Specifications	Bus cycle : 15ns (when XIN=66.6MHz) Logical address space : 4 Gbyte linear Internal data bus : 128 bit External data bus : 16 bit External address bus : 24 bit
Register Set	16 32-bit general registers 5 32-bit control registers
Instruction Set	83 basic instructions 3 addressing modes 16-bit/32-bit length instruction formats

1.3.2 Internal Memories

Table 1.2 M32R Internal Memory Specifications

Device Name	Specification
On-chip DRAM	2-Mbyte (16-Mbit) DRAM
Cache Memory	2-Kbyte SRAM which supports the following 3 modes of the memory controller : • On-chip DRAM space shared cache mode • External user space instruction cache mode • Cache-off mode

1.3.3 Internal Peripheral I/O Devices

Table 1.3 M32R Internal Peripheral I/O Device Specifications

Device Name	Specification
Memory Controller	DRAM access control
	DRAM refresh control
	Cache control
Bus Interface Unit (BIU)	128-bit buffer for burst transfer
Programmable Port	2 I/O ports

1.4 Pin Configuration

Figure 1.2 shows the M32R pin configuration.

||||| **NOTE** |||||

Precautions when connecting pins : In M32R, DO bit of the data bus is the MSB and bit D15 is the LSB. Bit A8 of the address bus will also be on the MSB side and bit A30 on the LSB side.

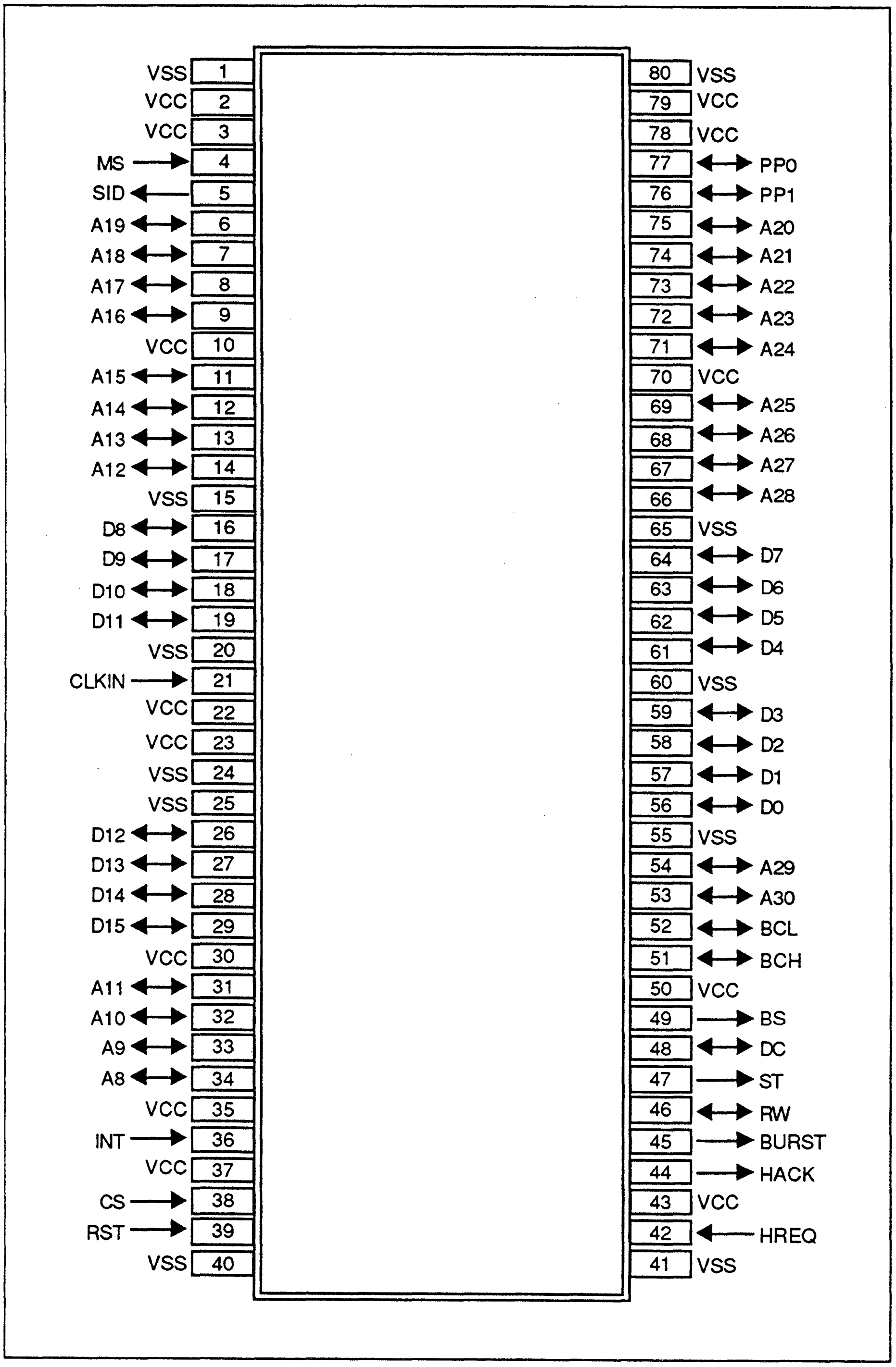


Figure 1.2 M32R Pin Configuration

1.5 Pin Function Descriptions

The following pin function descriptions are for M32R. A line on a symbol (e.g., $\overline{RST}$) means that the signal is active LOW ("L").

Table 1.4 Pin Description (1/4)

Name	Symbol	IN/OUT	Function
Clock	CLKIN	IN	The CLKIN(IN) pin of M32R (The CPU and internal memory system operate at four times the CLKIN signal frequency.)
Reset	$\overline{RST}$	IN	The Pin for inputting reset signals (If reset is set to "L", the M32R will start the reset operation, If reset is then set to "H", the reset state is cleared and the program is executed from the address set in the reset vector.)
Address Bus	A8-A30	INOUT	The 24-bit address (A8 to A31) for addressing a 16M-byte memory There are no outputs from external pins to least significant A31. During the write cycle, the byte position of the 16 bit data bus for effective write is $\overline{BCH}$, $\overline{BCL}$. The 16-bit data bus will be read during each read cycle. However, only the data available on the byte position will be transferred inside M32R. Since the address pin is bidirectional, the input the address from the system bus occurs when accessing the on-chip DRAM during the hold state of the M32R.
Space Identifier	SID	OUT	The space identifiers output distinguishes between user space and I/O space. user space : SID=0 I/O space : SID=1

Table 1.4 Pin Description (2/4)

Name	Symbol	IN/OUT	Function
Byte Control	$\overline{BCH}$, $\overline{BCL}$	INOUT	Indicates byte position to which significant data is transferred during the external bus cycle $\overline{BCH}$ indicates the high order byte (MSB) and $\overline{BCL}$ to the low order byte (LSB). $\overline{BCH}$ and $\overline{BCL}$ are both asserted during the read bus cycle in the M32R and in the write bus cycle, $\overline{BCH}$ and $\overline{BCL}$ are asserted to "L" in relation to the byte requesting write. Input byte control from the system when accessing the on-chip DRAM from outside the M32R.
Data Bus	D0-D15	INOUT	16-Bit data bus for use with external devices
Bus Start	$\overline{BS}$	OUT	When the M32R starts a bus cycle for an external bus, the $\overline{BS}$ pin is "L" when the bus cycle starts. When executing a burst transfer, $\overline{BS}$ is "L" when accessing the on-chip DRAM space or on-chip resources such as the on-chip I/O register.
Bus Status	ST	OUT	A signal to indicate that the bus cycle being started is an instruction read access or an operand read / write access, For instruction fetch access : ST=0 For operand access : ST=1 For hold and idle : ST= Unknown
Read/Write	R/$\overline{W}$	INOUT	The R/$\overline{W}$ signal to indicates read or write for external bus cycles. Input R/$\overline{W}$ from the system bus when accessing the on-chip DRAM from the external bus master. Read bus cycle : R/$\overline{W}$=1 Write bus cycle : R/$\overline{W}$=0

Table 1.4 Pin Description (3/4)

Name	Symbol	IN/OUT	Function
Burst	$\overline{\text{BURST}}$	INOUT	The M32R starts two consecutive bus cycle to access the 32-bit aligned data with the 32-bit boundary. In the external instruction cache mode, bus cycles are started consecutively eight times for reading the 128-bit size aligned data along the 128-bit boundary for cache replacement. The $\overline{\text{BURST}}$ pin is asserted to "L" during the continuous bus cycle. The address when carrying out 32-bit size data access will be output in the order of high order 16-bit (MSB) to low order 16 bits (LSB). The address, when carrying out 128-bit size data access, is output for each access cycle in wrap around form within the 128-bit alignment boundary with an optional 16-bit address in the lead.
Data Complete	$\overline{\text{DC}}$	INOUT	When the M32R starts an external bus cycle, the wait cycle will be automatically inserted until the $\overline{\text{DC}}$ pin asserts from the bus slave on the system bus. Wait control from $\overline{\text{DC}}$ is also effective even in the bus cycle during a burst transfer. The $\overline{\text{DC}}$ pin is bi-directional and when the on-chip DRAM is being accessed by an external bus master when the M32R is in the hold state, the M32R asserts the $\overline{\text{DC}}$ pin and notifies the system bus that the bus cycle to the on-chip DRAM is complete.
Hold Request	$\overline{\text{HREQ}}$	IN	An input pin that requests bus rights of the system bus the M32R converts to HOLD state when $\overline{\text{HREQ}}$ is asserted to "L".
Hold Acknowledge	$\overline{\text{HACK}}$	OUT	The pin to notify that the M32R has converted to the hold state and has relinquished bus rights on the system bus.

Table 1.4 Pin Description (4/4)

Name	Symbol	IN/OUT	Function
Chip Selector	$\overline{CS}$	IN	The pin for demanding access to the on-chip DRAM from the outside with M32R in the hold state If "L" is asserted, the M32R accesses the on-chip DRAM with the address input from the address pin.
External Interrupt	$\overline{INT}$	IN	The pin for inputting an external interrupt
Master/Slave	$M/\overline{S}$	IN	The pin to set the M32R default operation on the system bus to bus master or the bus slave. If set to bus slave, the M32R will not assert vector fetch when reset is cleared.
Programmable Port	PP0,PP1	INOUT	A 2-bit programmable I/O port
Power, Ground	VCC, VSS	IN	The power pins and ground pins of the M32R power supply. Connect all multiple VCC and VSS pins to the power supply and ground.

1.6 Address Space

1.6.1 Logical Address and Physical Address

The logical address is always handled at 32 bit width and has a 4Gbyte linear space. This address space is divided into two spaces by the most significant bit (Space Identifier SID) of the logical address.

In the M32R, SID and the low order 24 bits actually become effective relative to the 32-bit logical address.

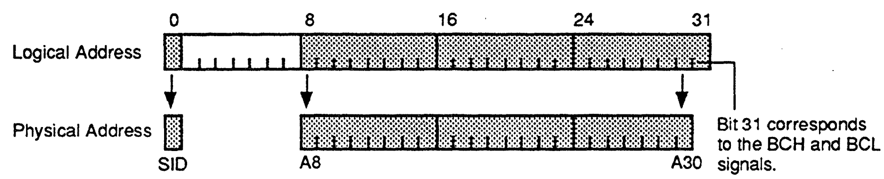


Figure 1.3 Logical Address and Physical Address

1.6.2 User Space (SID=0)

SID = 0 (H'00000000 - H'7FFFFFFF) is the user space. The usable space is the 16M bytes of the H'00000000 - H'00FFFFFF address.

- On-chip DRAM Space : 2Mbytes of address (H'00000000 - H'001FFFFF) is the on-chip DRAM space. Of this, the H'00000000 - H'00000047 addresses are used as the EIT vector area.

- External Space : 14Mbytes of address (H'00200000 - H'FFFFFFF) is the external space. Outputs signals are required for accessing external devices. The last 16-byte area (H'FFFFFFF0 - H'FFFFFFFF) is used as the EIT vector area (reset vector entry).

1.6.3 I/O Space (SID=1)

SID = 1 (H'80000000 - H'FFFFFFFF) is the I/O space. Of this, the usable space is 16Mbytes of address (H'FF000000 - H'FFFFFFFF). There is no caching of the I/O space regardless of the Cache Mode.

- I/O Space for User — The 8Mbytes of the H'FF000000 - H'FF7FFFFF addresses is the I/O space for user. Outputs signals are necessary for accessing external devices.
- I/O Space for ICE — The 4Mbytes of the H'FF800000 - H'FFBFFFFF address is the system space. This is used development tools for in-circuit emulators and debugging monitors. To assure operation as a development tool, these addresses must not be used.
- I/O Space forInternal I/O Regis ters
  The 4Mbytes of the H'FFC00000 - H'FFFFFFFF addresses is the I/O space for internal. Registers for the on-chip memory and I/O ports are located here.

1.6.4 Memory Mapping

Figure 1.4 shows the memory mapping of the M32R. Figure 1.5 shows the mapping of EIT vectors and internal I/O registers.

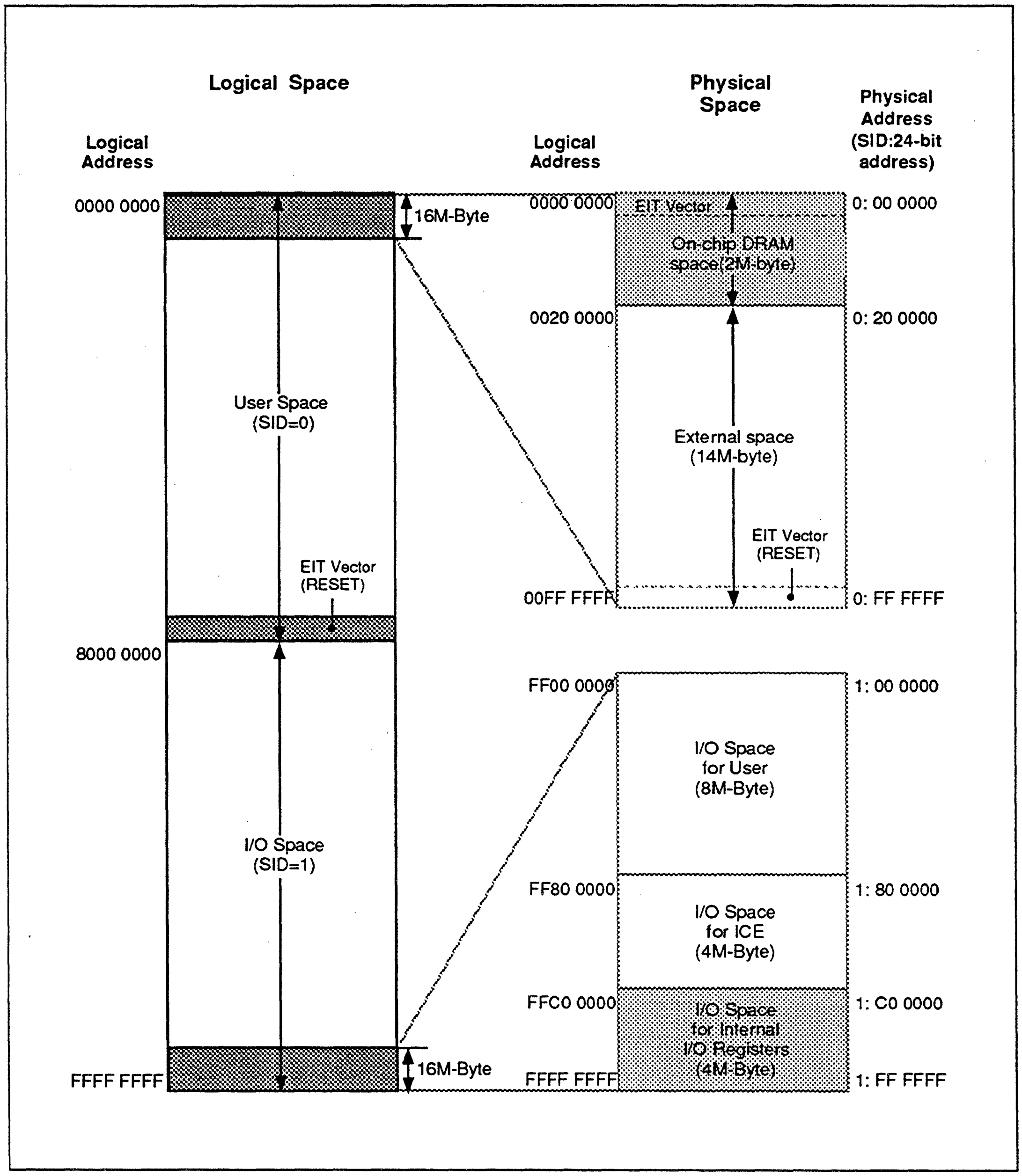


Figure 1.4 Mapping of Memory

EIT Vector Space

Logical Address		Physical Address (SID:24-bit address)
0000 0000	Vector Entry for TRAP0	0:00 0000
0000 0004	Vector Entry for TRAP1	0:00 0004
0000 0008	Vector Entry for TRAP2	0:00 0008
0000 000C	Vector Entry for TRAP3	0:00 000C
0000 0010	Vector Entry for TRAP4	0:00 0010
0000 0014	Vector Entry for TRAP5	0:00 0014
0000 0018	Vector Entry for TRAP6	0:00 0018
0000 001C	Vector Entry for TRAP7	0:00 001C
0000 0020	Vector Entry for TRAP8	0:00 0020
0000 0024	Vector Entry for TRAP9	0:00 0024
0000 0028	Vector Entry for TRAP10	0:00 0028
0000 002C	Vector Entry for TRAP11	0:00 002C
0000 0030	Vector Entry for TRAP12	0:00 0030
0000 0034	Vector Entry for TRAP13	0:00 0034
0000 0038	Vector Entry for TRAP14	0:00 0038
0000 003C	Vector Entry for TRAP15	0:00 003C
0000 0040	Vector Entry for AE	0:00 0040
0000 0044	Vector Entry for EI	0:00 0044
7FFF FFF0	Reset Vector Entry	0:FF FFF0
7FFF FFF4		0:FF FFF4
7FFF FFF8		0:FF FFF8
7FFF FFFC		0:FF FFFC

Internal I/O Registers

Logical Address		Physical Address
FFC0 0000	Reserved	1:C0 0000
FFFF FFE0	PPCR0	1:FF FFE0
FFFF FFE4	PPCR1	1:FF FFE4
FFFF FFE8	PPDR0	1:FF FFE8
FFFF FFEC	PPDR1	1:FF FFEC
FFFF FFF0		1:FF FFF0
FFFF FFF4	MSPR	1:FF FFF4
FFFF FFF8	MDRR	1:FF FFF8
FFFF FFFC	MCCR	1:FF FFFC

Figure 1.5 Mapping of EIT Vector and Internal I/O Register

2

Programming Model

2.1 Registers

The M32R CPU has 16 general purpose registers, five control registers, an accumulator and a program counter.

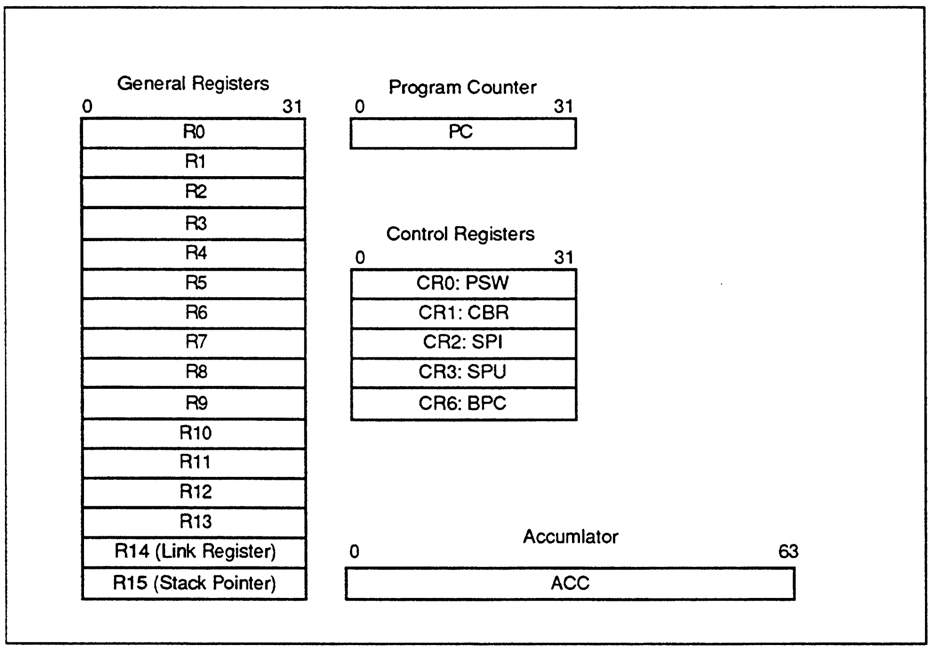


Figure 2.1 M32R Register Set

2.1.1 General Registers R0-R15

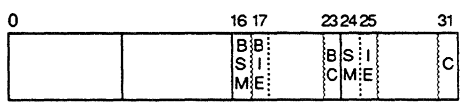
The general purpose register is 32 bits wide with 16 registers (R0-R15) and is used to hold data and base addresses. R14 is used as a link register and R15 as a stack pointer (SPI or SPU). The interrupt stack pointer (SPI) and the user stack pointer (SPU) change over corresponding to the stack mode (SM) bit value of processor status word (PSW).

2.1.2 Control Registers

There are five control registers consisting of the processor status word register (PSW), condition bit register (C), interrupt stack pointer (SPI), user stack pointer (SPU) and backup PC (BPC). MVTC and MVFC instructions are used for setting and reading the control register.

2.1.2.1 Processor Status Word Register, PSW (CRO)

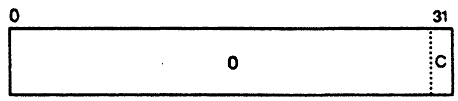
The PSW register contains the stack mode (SM), interrupt enable (IE), condition bit (C), and also the backup stack mode (BSM), backup interrupt enable (BIE), backup condition bit (BC) for saving the three bits.



[Bit]	Symbol	Name and Function
[16]	BSM	Backup Stack Mode Sets value of SM bit when EIT occurs
[17]	BIE	Backup Interrupt Enable Sets value of IE bit when EIT occurs
[23]	BC	Backup Condition Bit Sets value of C bit when EIT occurs
[24]	SM	Stack Mode 0 : Use the interrupt stack pointer 1 : Use the user stack pointer
[25]	IE	Interrupt Enable 0 : Does not accept interrupt 1 : Accepts interrupt
[31]	C	Condition Bit Indicates carry, borrow and overflow of operation results corresponding to the instructions
All other bits		0 (fixed)

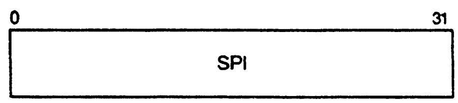
2.1.2.2 Condition Bit Register, CBR (CR1)

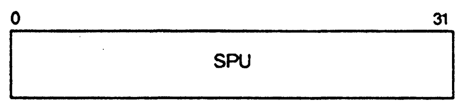
The C bit from PSW is contained in CBR, a separate register. The values of the C bit of PSW will be reflected in the CBR register. Read only is possible of this register. Attempts to write to CBR with MVTC instructions will be ignored.



2.1.2.3 Interrupt Stack Pointer, SPI (CR2) User Stack Pointer, SPU (CR3)

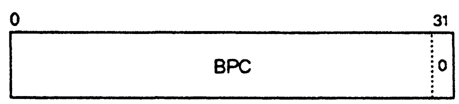
The SPI and SPU can be accessed as general purpose register R15. Use of SPI or SPU will be determined by the SM bit of PSW.





2.1.2.4 Backup PC, BPC (CR6)

BPC is the register which contains the PC value when EIT occurs. Bit 31 will be fixed at 0.



2.1.3 Accumulator, ACC

ACC is a 64-bit register used for DSP functions (refer to 2.4.5).

Use the MVTACHI and MVTACLO instructions for setting the accumulator. The high order 32 bits (bits 0-31) can be set with the MVTACHI instruction and the low order 32 bits (bits 32-63) can be set with the MVTACLO instruction.

Use the MVFACHI, MVFACLO and MVFACMI instructions for read operations. The high order 32 bits (bits 0-31) are read with the MVFACHI instruction, the low order 32 bits (bits 32-63) with the MVFACLO instruction and the middle 32 bits (bits 16-47) with the MVFACMI instruction.

Also, since the accumulator is also being used to execute the MUL instruction, care will be required as its value will be indeterminate at this time.

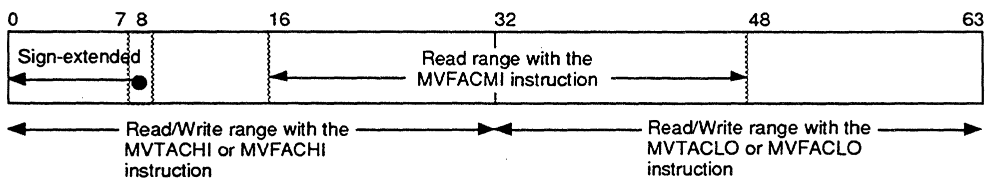
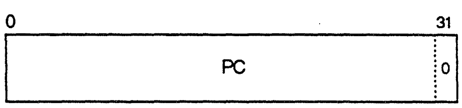


Figure 2.2 Accumulator

The sign-extended code value of bit 8 will always be read for bits 0 to 7. A write operation in this section will be ignored.

2.1.4 Program Counter, PC

PC is a 32-bit counter that holds the address of the instruction being executed. Since M32R instructions commence with even-numbered addresses, the LSB (Bit 31) becomes 0.



2.2 Data Formats

2.2.1 Data Types

Signed and unsigned integers of bytes (8-bit), halfwords (16-bit), and words (32-bit) are supported as data in the M32R instruction set. A signed integer is represented in a 2's complement format.

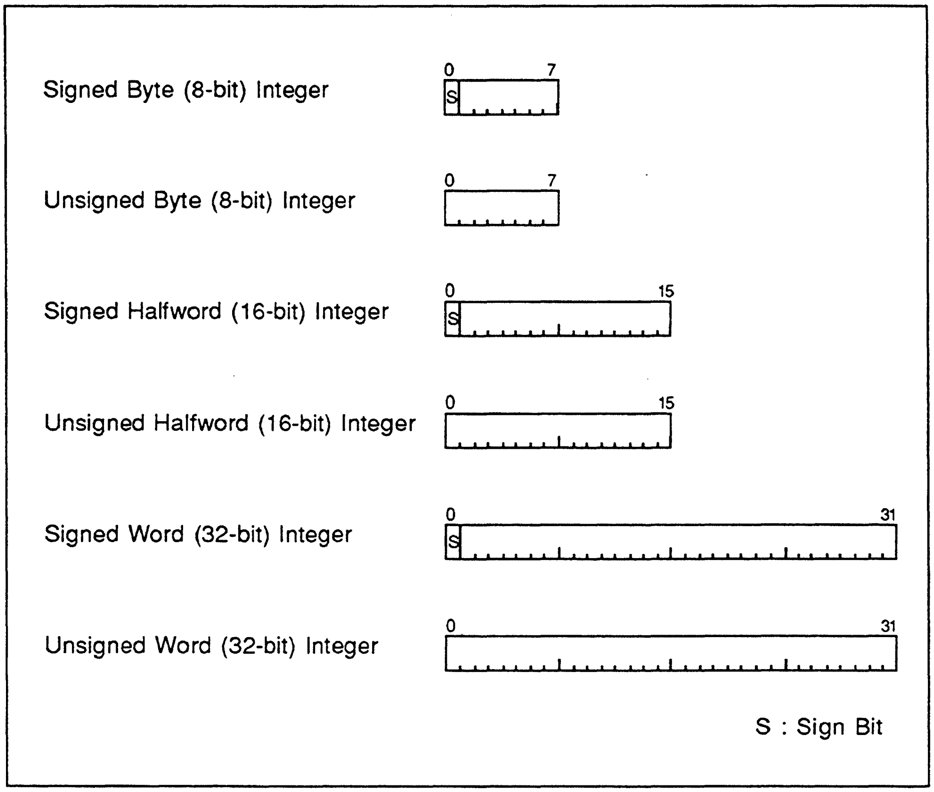
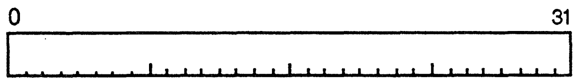


Figure 2.3 Data Type

2.2.2 Data Format in a Register

Data size in a register is always a full word (32 bits). A byte (8 bits) and halfword (16 bits) data of the memory are sign-extended to a 32 bit code or zero-extended and loaded into the register.



2.2.3 Data Format in The Memory

The data in the memory consist of the three types : byte (8 bits), halfword (16 bits) or word (32 bits).

Although the byte data can be located in any address, the halfword data and word data are located on the halfword boundary and word boundary respectively. If an attempt is made to access memory data not on the boundary, an address exception will occur.

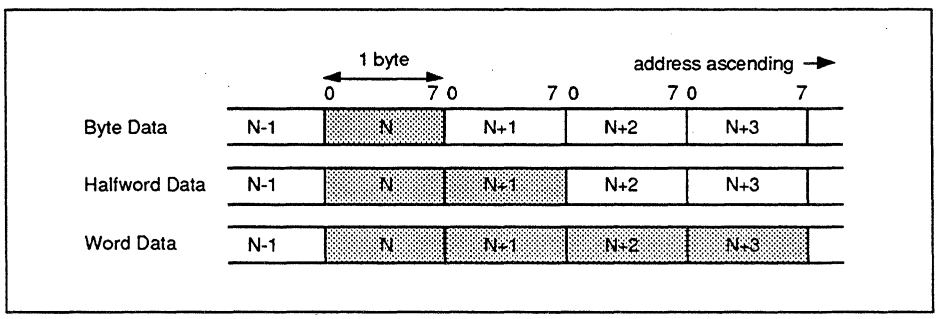


Figure 2.4 Bytes, halfwords, and words in memory (When address N is specified)

2.3 Addressing Modes

The addressing mode is the mode in which the M32R instructions specify data to be processed. The descriptive operands are determined by the addressing mode. The descriptive operands are shown in square brackets ([ ]) with " Rn" meaning a general register, "CRn" a control register, "disp" displacement and " *n* " the number of bits. The M32R supports the following addressing modes :

- Register direct [Rn or CRn]

  The specified register value becomes the effective data. Furthermore, the control register can only be specified by the MVTC instruction or the MVFC instruction.

- Register indirect [@Rn]

  Data stored in the memory location indicated by the specified register value becomes the effective data. This mode can be used by all load/store instructions.

- Register relative indirect [@(disp16, Rn)]

  Data stored in the memory location indicated by the address from the following computation becomes the data :
  (register value) + better to break line here
  (32-bit sign-extended value of 16-bit integer)

- Register indirect + Register update

  There are 3 types as follows :

  1. [@Rn+] Data stored in the memory ocation indicated by the specified register before updating is the data.
     After referencing the register, 4 will be added to the register value. This mode can be used with the load instructions.

  2. [@+Rn] Data stored in the memory location indicated by the specified register after updating becomes the data.
     Before referencing the register, 4 will be added to the register value. This mode can be used with the store instructions.

  3. [@-Rn] Data stored in the memory location indicated by the specified register after updating becomes the data.

Before referencing the register, 4 will be deducted from the register value.

- Immediate [imm *n*] — The effective data is the 32-bit sign-extended or zero-extended value of *n*-bit immediate value.

- PC relative indirect [disp *n*]
  The effective data is in the location (address) as follows :
  (PC value) + (2bit left-shifted value with a 32-bit sign extended *n*-bit immediate value)

2.4 Instruction Formats

There are two major instruction formats : two 16-bit instructions packed together in a word, and a single 32-bit instruction.

16-bit length instruction (A)	16-bit length instruction(B)

32-bit length instruction

The most significant bit of a 32-bit instruction is always 1.

The most significant bit of a 16-bit instruction placed the upper halfword (i.e. the instruction (A) in the above figure), is always 0.

The most significant bit of the 16-bit instruction in the lower halfword (B) indicates execution sequence of the instructions. If it is 0, the instructions (A) and (B) are executed sequentially; (B) is executed after (A). If the most significant bit of the instruction (B) is 1, the instructions (A) and (B) are executed in parallel.

0	(A)	0	(B)	Sequential execution (A) -> (B)
0	(A)	1	(B)	Parallel execution (A) and (B)
1	32-bit length instruction			

While any instruction which can be placed in (A) can also be placed in (B) for sequential execution, the current implementation allows only the instruction, NOP, as the instruction (B) for parallel execution. The code is as follows :

0	(A)	1111	0000	0000	0000	Parallel execution (A) and NOP

In Appendix A, Detail Description of Instructions, a 16-bit instruction encoding is shown as it is placed in (A); the most significant bit is 0.

The instruction encoding is as follows :

- 16-bit instruction :

op1	R1	op2	R2	R1 = R1 *op* R2
op1	R1	c		R1 = R1 *op* c
op1	cond	c		Branch (short displacement)

- 32-bit instruction :

op1	R1	op2	R2	c	R1 = R2 *op* c
op1	R1	op2	R2	c	Compare and Branch
op1	R1	c			R1 = R1 *op* c
op1	cond	c			Branch

2.5 Instruction Set

2.5.1 Load/Store Instructions

The load/store instructions carry out data transfers between the register and memory.

The load/store instructions are listed in Table 2.1.

Table 2.1 Load/Store instructions

Instruction	Function
LDB	Load byte
LDUB	Load byte unsigned
LDH	Load halfword
LDUH	Load halfword unsigned
LD	Load
LOCK	Load locked
STB	Store byte
STH	Store halfword
ST	Store
UNLOCK	Store unlocked

The are there types of Addressing modes (Refer to section 2.3 for details on the addressing modes) :

- Register indirect Can be specified by all load/store instructions
- Register relative indirect
  Can be specified by instructions other than LOCK, UNLOCK
- Register indirect + Register update
  @Rn+ can be specified with LD, LDB, LDUB, LDH, or LDUH
  @+Rn can be specified with ST, STB, or STH
  @-Rn can be specified with ST, STB, or STH

For access size, 3 data types, single byte(8bits), halfword(16bits), and word(32bits) can be used. The data position in the memory specified by access size and the low order 2 bits of the address, is as shown in Figure 2.5.

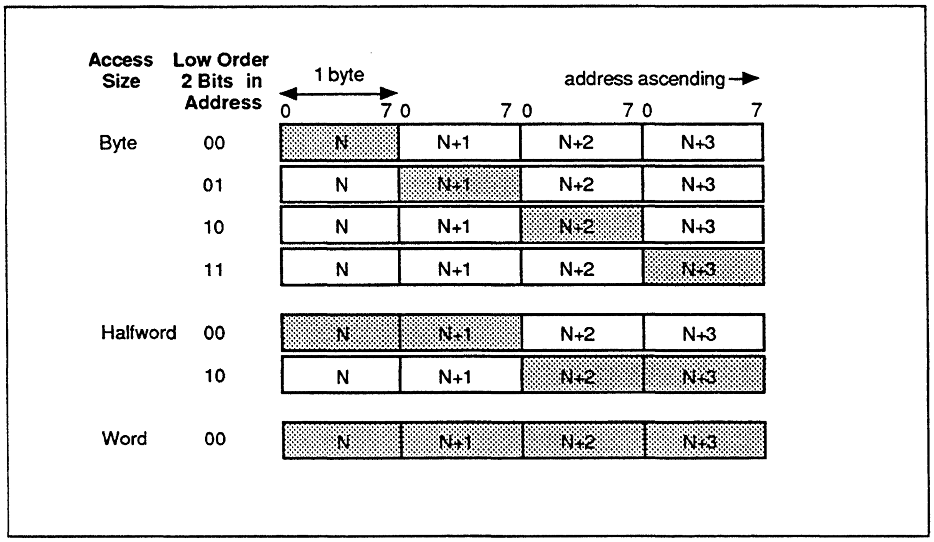


Figure 2.5 Data Position in Memory

When accessing halfword and word size data, it is necessary to specify the address including halfword alignment or word alignment. In other words, halfword size must be such that the low order 2 bits of the address be 00 or 10, and word size be such that the low order 2 bits of the address be 00. If an unaligned address is specified, address exception will occur.

When accessing byte data or halfword data with load instructions, it is stored as 32 bit data with high order bits sign-extended or zero-extended.

2.5.2 Arithmetic Instructions

The arithmetic instructions carry out transfer, comparison, arithmetic operation, logic operation, multiply and divide and shift operation between registers or immediate values to a register.

The arithmetic instructions are listed Table 2.2.

Table 2.2 Arithmetic Instructions (1/2)

Group	Instruction	Function
Transfer	LDI	Load immediate
	LD24	Load 24-bit immediate
	SETH	Set high-order 16-bit
	MV	Move register
	MVFC	Move from control register
	MVTC	Move to control register
Compare	CMP	Compare
	CMPI	Compare immediate
	CMPU	Compare unsigned
	CMPUI	Compare unsigned immediate
Arithmetic operation		
	ADD	Add
	ADD3	Add 3-operand
	ADDV	Add (with overflow checking)
	ADDV3	Add 3-operand
	ADDI	Add immediate
	ADDX	Add with carry
	SUB	Subtract
	SUBV	Subtract (with overflow checking)
	SUBX	Subtract with borrow
	NEG	Negate
	MUL	Multiply
	DIV	Divide
	DIVU	Divide unsigned
	REM	Remainder
	REMU	Remainder unsigned
Logic operation	AND	AND
	AND3	AND 3-operand
	OR	OR
	OR3	OR 3-operand
	XOR	Exclusive OR
	XOR3	Exclusive OR 3-operand
	NOT	Logical NOT

Table 2.2 Arithmetic Instructions (2/2)

Group	Instruction	Function
Shift	SLL	Shift left logical
	SLLI	Shift left logical immediate
	SLL3	shift left logical 3-operand
	SRL	Shift right logical
	SRLI	Shift right logical immediate
	SRL3	Shift right logical 3-operand
	SRA	Shift right arithmetic
	SRAI	Shift right arithmetic immediate
	SRA3	Shift right arithmetic 3-operand

2.5.3 Branch Instructions

The branch instructions are used to change program flow.

The branch instructions are listed in Table 2.3.

Table 2.3 Branch instructions

Instruction	Function
JMP	Jump
JL	Jump and link
BRA	Branch
BL	Branch and link
BC	Branch on C-bit
BNC	Branch on not C-bit
BEQ	Branch on equal
BNE	Branch on not equal
BEQZ	Branch on equal zero
BNEZ	Branch on not equal zero
BLTZ	Branch on less than zero
BGEZ	Branch on greater than or equal zero
BLEZ	Branch on less than or equal zero
BGTZ	Branch on greater than zero
NOP	No operation

The only address that can be specified at the branches is a word- aligned address. For example, branching to B, D and H instruction is not possible as shown in Figure 2.6.

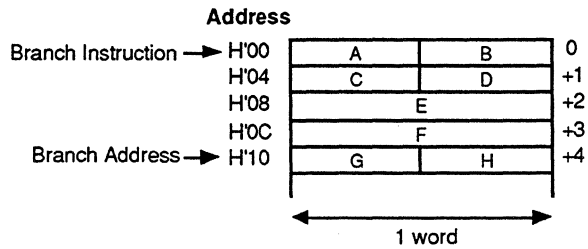


Figure 2.6 Branch Address

In the JMP and JL instructions, the register value becomes the branch address. However, the low-order 2-bit value of the register will be ignored. Other branch instructions have branch addresses as follows :

(PC value of branch instruction) + (2-bit shifted value with sign-extended immediate value on the left)

However, the low order 2-bit of the address value becomes 00 when carrying out addition, For example, when branching to the G instruction in the above diagram, the immediate value will become 4 for either A or B instruction.

The BRA, BL, BC and BNC instructions can specify immediate of 8-bit or 24-bit addresses . BEQ, BNE, BEQZ, BNEZ, BLTZ, BGEZ, BLEZ, and BGTZ instructions can specify an immediate of 16-bit.

Simultaneous with branching of the JL or BL instructions for subroutine calls, the return address PC value is stored in R14. The low order 2 bits of the address value stored in R14 (PC value of the branch instruction + 4) are always cleared to zero. For example, both instructions A and B return to instruction C (See Figure 2.6).

2.5.4 OS-Related Instructions

The OS-related instructions trap and carry out restoration from EIT.

The OS-related instructions are listed in Table 2.4.

Table 2.4 OS-related instructions

Instruction	Function
TRAP	Trap
RTE	Return from EIT

2.5.5 DSP Function Instructions

The DSP function instructions carry out multiplication and sum of the operation of 32 bits X 16 bits and 16 bits X 16 bits. Then also round off data in the accumulator and carry out transfer of data between the accumulator and general purpose register.

The DSP function instructions are listed in Table 2.5.

Table 2.5 DSP Function instructions

Instruction	Function
MULHI	Multiply high-order halfwords
MULLO	Multiply low-order halfwords
MULWHI	Multiply word and high-order halfword
MULWLO	Multiply word and low-order halfword
MACHI	Multiply-accumulate high-order halfwords
MACLO	Multiply-accumulate low-order halfwords
MACWHI	Multiply-accumulate word and high-order halfword
MACWLO	Multiply-accumulate word and low-order halfword
RACH	Round accumulator halfword
RAC	Round accumulator
MVFACHI	Move from accumulator high-order word
MVFACLO	Move from accumulator low-order word
MVFACMI	Move from accumulator middle-order word
MVTACHI	Move to accumulator high-order word
MVTACLO	Move to accumulator low-order word

3

Exception,Interrupt,Trap (EIT)

3.1 About EIT

There are special cases such that an error occurs when executing a program. The M32R suspends execution of the program and executes a different program. These special cases are called EIT (Exception, Interrupt, Trap) and consist of the following :

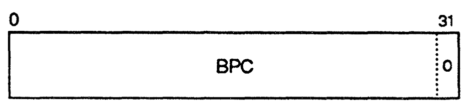
- Exception — Exception relates to the program being executed and occurs when an instruction cannot be executed normally. The address exception (AE) belongs to this category.
  - Address Exception (AE) — AE occurs when attemption to access with an unaligned address using the load/store instructions.
- Interrupt — Interrupt is unrelated to the program being executed and occurs with the input of an external signal to the M32R. The reset interrupt (RI) and external interrupt (EI) belong to this category.
  - Reset Interrupt (RI) — RI occurs when asserting the RESET signal.
  - External Interrupt (EI) — EI occurs when asserting the INT signal.
- Trap — The software interrupt is generated by the TRAP instruction in an user program, etc. (e.g. system call for the operating system).

• Trap (TRAP) Trap occurs when the TRAP instruction is executed.

3.2 EIT-Related Control Registers

3.2.1 BPC

The BPC register is used to save the PC value when an EIT occurs. If an EIT is received, the PC value will be automatically set in the BPC. The BPC value will be returned to PC by executing the RTE instruction. However, the low-order 2 bits of the returning PC will become zero regardless of the value of BPC. In other words, the value of the returned PC will always be on the 32 bit boundary.



3.2.2 PSW

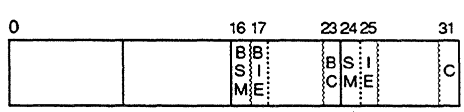
The BSM, BIE and BC bits are used for saving the SM, IE and C bits, respectively. When an EIT is received, the SM, IE and C bit values are updated after SM, IE and C bits are automatically copied into the BSM, BIE and BC bits, respectively.

By executing the RTE instruction, the BSM, BIE and BC bits will automatically be restored to the SM, IE and BC bits, respectively.

The SM bit specifies the stack pointer to be used. The interrupt stack pointer (SPI) will be used when SM=0 and user stack pointer (SPU) will be used when SM=1.

If an interrupt is accepted, the SM bit will be automatically set to zero and will specify the stack mode for the interrupt.

The IE bit controls the permitting/forbidding of an interrupt. The interrupt is accepted when IE=1 and is forbidden when IE=0. If an EIT is accepted, the IE bit is automatically set to zero creating an interrupt inhibit state. The C bit is also set to zero if an EIT is accepted.



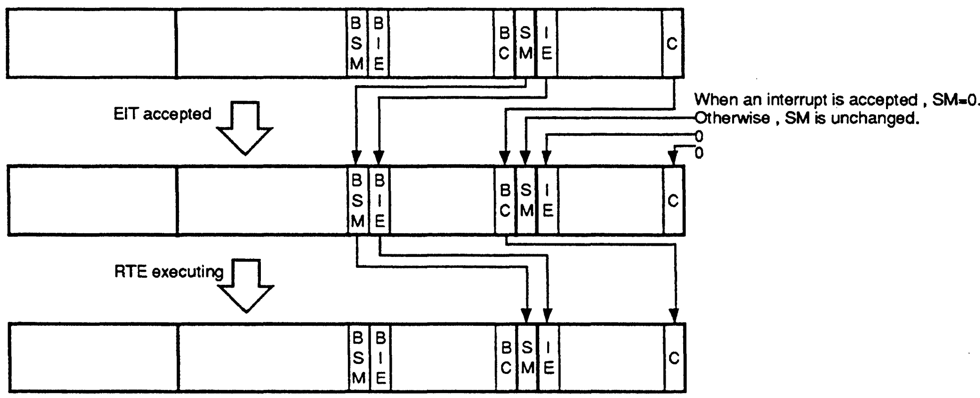


Figure 3.1 Saving and Restoring PSW

3.3 EIT Processing

The EIT processing consists of the part being automatically processed by hardware and the part being processed by a program (EIT handler). The EIT processing is shown below (except for the case of the reset interrupt) :

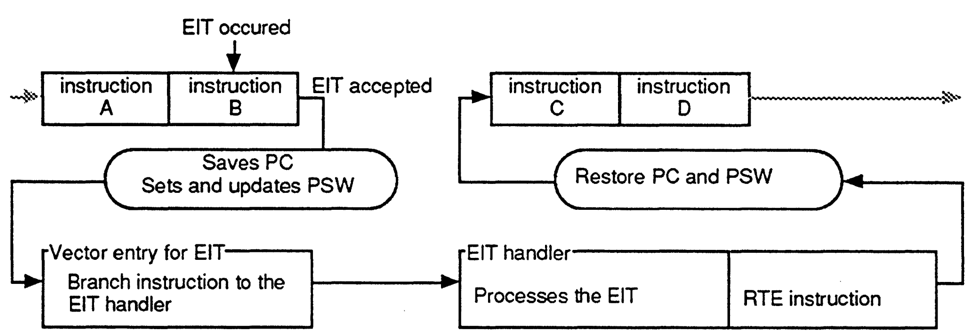


Figure 3.2 EIT Processing Image (Except RI)

With the M32R, the processing jumps to the EIT vector after saving PC and PSW if the EIT is accepted. In the EIT vector, the EIT vector entry of one word (4 words when resetting) is allocated for each EIT and branch instruction by the EIT handler. Finally, if processing by the EIT handler is completed, reversion from the EIT processing is possible by executing an RTE instruction.

The following steps are carried out automatically in the M32R.

(1) Before jumping to the EIT vector and after accepting EIT

- Save PC — The M32R saves the contents of PC, the return site, in BPC.

- Save and update PSW

  The M32R saves the SM, IE and C bits of PSW in the BSM, BIE and BC bits, respectively.

  The SM bit of PSW is set to zero in the case of interrupt (RI, EI) and becomes the mode in which the interrupt stack pointer (SPI) is used. The SM bit changes only in the case of interrupt.

  The IE bit of PSW will be set to zero and the interrupt will be inhibited. The C bit of PSW is set to zero.

- Jump to EIT vector

  The M32R jumps to the EIT vector after completing the saving of PC and PSW.

(2) Reset by RTE instruction in EIT processing

- Restore PSW — The values contained the BSM, BIE and BC bits of PSW are restored respectively in the SM, IE and C bits.

BSM	unchanged
BIE	unchanged
BC	unchanged
SM	saved value in BSM is restored in SM
IE	saved value in BIE is restored in IE
C	saved value in BC is restored in C

- Restore PC — The M32R sets the BPC value in PC. The two low order bits of PC are set to zero. The jump destination of the RTE instruction always becomes a word boundary. The BPC value remains unchanged.

3.4 Priority of EITs

When two or more EITs occur at the same time, a higher priority EIT is accepted first. The priority of the EITs is summarized in Table 3.1.

Table 3.1 Priority rank of EIT

Priority	EIT	Instruction Processing is
Highest	Reset Interrupt (RI)	Abandoned
↑	Address Exception (AE)	Canceled
↓	Trap (TRAP)	Completed
Lowest	External Interrupt (EI)	Completed

3.5 EIT Vector Entry

The EIT vector entries are in the user space (SID=0). Refer to section 1.6, "Address Space".

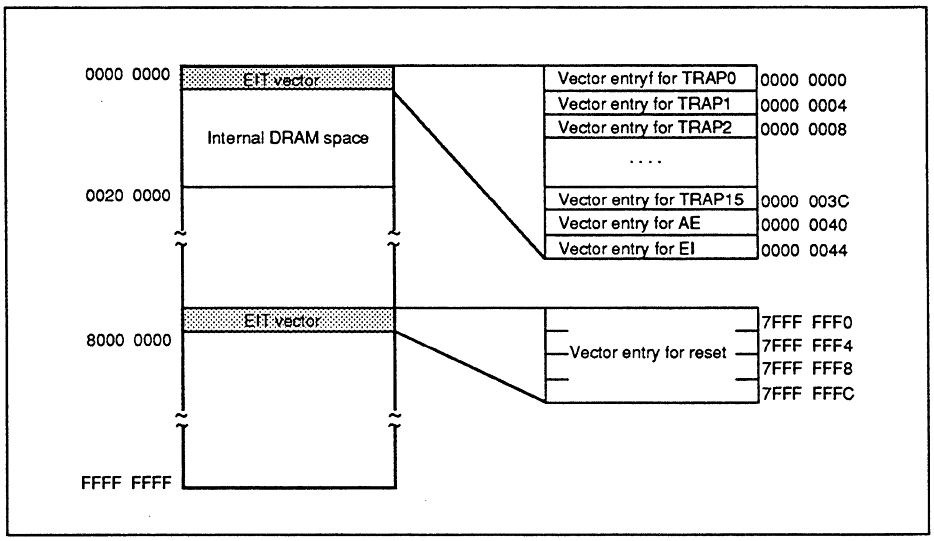


Figure 3.3 EIT Vector

3.6 Detailed Description of each EIT

3.6.1 Table of each EIT Processing

Table 3.2 EIT Processing

Category	Symbol	Trap No.	Saved Value to BCP	Contents of PSW ("-" means "unchanged") BSM	BIE	BC	SM	IE	C	Address of EIT Vector Entry
Trap	TRAP	0	(PC for the TRAP instruction) + 4	SM	IE	C	-	0	0	H'00000000
		1								H'00000004
		2								H'00000008
		3								H'0000000C
		4								H'00000010
		5								H'00000014
		6								H'00000018
		7								H'0000001C
		8								H'00000020
		9								H'00000024
		10								H'00000028
		11								H'0000002C
		12								H'00000030
		13								H'00000034
		14								H'00000038
		15								H'0000003C
Address Exception	AE		PC for the last AE instruction	SM	IE	C	-	0	0	H'00000040
External Interrupt	EI		PC for next instruction	SM	IE	C	0	0	0	H'00000044
Reset Interrupt	RI		unfixed	unfixed	unfixed	unfixed	0	0	0	H'7FFFFFF0

3.6.2 Reset Interrupt (RI)

- Starting condition

RI is always permitted with the assert of RESET signal. The reset interrupt is permitted with the highest priority.

- EIT Processing

• Saving PC

BPC becomes an unspecified value.

• Updating PSW

Each bit of PSW will be updated to the following values :

Bit	Value
BSM	unspecified
BIE	unspecified
BC	unspecified
SM	0
IE	0
C	0

• Jumping to the EIT vector entry

The M32R jumps to location H'7FFFFFF0.
For reset sequence, the program stored in 16-byte area beginning from H'7FFFFFF0 is executed. At end of this program, it should explicitly jump to proper address to initiate program.

3.6.3 Address Exception (AE)

- Starting condition

AI occurs when the load/store instructions attempt to access memory with an unaligned address.

Combinations of instruction and address with address exception starts are as follows :

• When low order 2 bits of the address is 01,11 in the LDH or STH instructions.

• When low order 2 bits of the address is 01, 10, 11 in the LD, ST, LOCK, UNLOCK instructions.

When the address exception occurs, there is no memory access with the instruction.

When an address exception is detected, the address exception is accepted even from an external interrupt request.

- EIT Processing

- Saving PC

  The PC value of instruction causing an address exception is set in BPC. For example, if the instruction causing address exception is in Address 4, the number 4 is stored in BPC. If Address 6, 6 is stored in BPC. However, care is required since the return destination with the RTE instruction becomes Address 4 in both cases (since the low order 2 bit is set to zero when returning to PC). In this case, the value of Bit 30 of BPC indicates whether the instruction causing address exception is on the word boundary (BPC[30]=0) or not (BPC[30]=1).

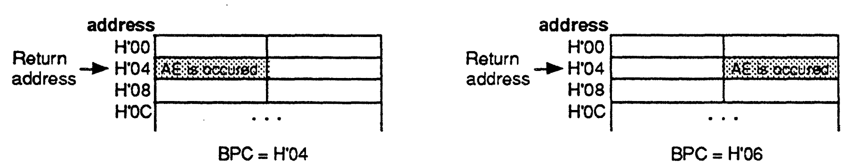


Figure 3.4 Return Destination of Address Exception

- Updating PSW

  Each bit of PSW will be updated to the following values :

Bit	Value
BSM	SM
BIE	IE
BC	C
SM	unchanged
IE	0
C	0

- Jumping to the EIT vector entry

  The M32R jumps to location H'00000040.

3.6.4 Trap (TRAP)

- Starting condition

  Starts by execution of the TRAP instruction. Even if there is an external interrupt request while executing a TRAP instruction, the trap is accepted.

- EIT Processing

  - Saving PC

    The value (PC value of TRAP instruction + 4) is stored in BPC.

    For example, 8 is stored in BPC with a TRAP instruction of Address 4. 10 is stored in BPC with the TRAP instruction of Address 6. However, care will be required with the RTE instruction since the return destination will be Address 8 in both cases (The tow low order bits are set to zero when returning to PC). In this case, the value of Bit 30 of BPC indicates whether the TRAP instruction is on the word boundary (BPC[30]=0) or not (BPC[30]=1).

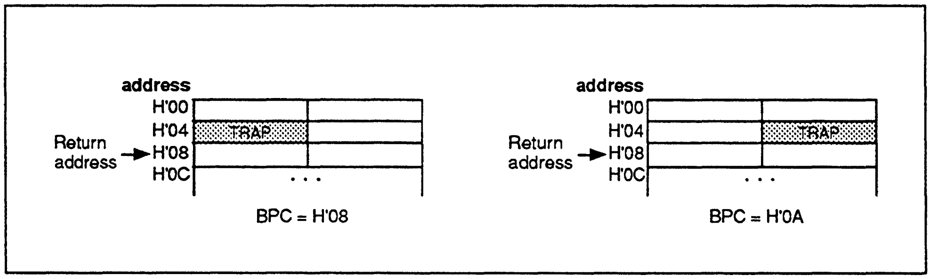


Figure 3.5 Return Destination of Trap

  - Updating PSW

    Each bit of PSW will be updated with the following values :

Bit	Value
BSM	SM
BIE	IE
BC	C
SM	unchanged
IE	0
C	0

• Jumping to the EIT vector entry

The M32R jumps to the (trap No. × 4) address.

Trap No.	Address of EIT Vector Entry
0	H'00000000
1	H'00000004
2	H'00000008
:	:
:	:
14	H'00000038
15	H'0000003C

3.6.5 External Interrupt (EI)

- Starting condition

Check the INT pin at the instruction interval on the word boundary. The INT pin is asserted at this time and external interrupt is accepted if the IE bit of PSW is 1. Therefore, there will be no external interrupts immediately after executing the 16 bit instruction on the word boundary.

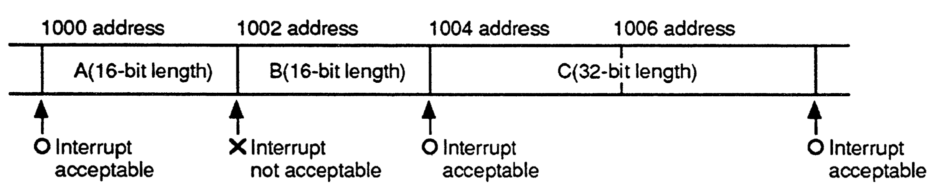


Figure 3.6 Timing of External Interrupt Accept

- EIT Processing

• Saving PC

The PC value of the next instruction to be executed is set in the BPC. If an external interrupt is received immediately after executing a branch instruction, the branch target address is set in the BPC. When an external interrupt is received immediately after executing a conditional branch instruction, the branch target address is set in the BPC if the branch is taken, the next instruction PC is set if the branch is not taken.

- Updating PSW

  Each bit of PSW will be updated to the following values :

Bit	Value
BSM	SM
BIE	IE
BC	C
SM	0
IE	0
C	0

- Jumping to the EIT vector entry

  The M32R jumps to location H'00000044.

4

Internal Memory System

4.1 Overview of Internal Memory System

The memory system of the M32R features the following :

- 2MB of high-speed DRAM, 2KB of high-speed SRAM as a cache, and the memory controller for DRAM and SRAM
- A 128-bit bus connecting the CPU, cache, and DRAM (This bus eliminates performance loss due to memory bus bottlenecks and allows the M32R CPU core to perform at its maximum speed.)
- A cache SRAM mode which can be switched as follows :

  Instruction and data cache for the internal DRAM
  Instruction cache for the external user space

4.2 Internal Memory System Configuration

The M32R/DRAM internal memory system is configured according to the setting of the memory controller configuration register, described later.

4.2.1 On-chip DRAM Space Shared Cache Mode

Figure 4.1 shows the configuration of the M32R internal memory system when selecting in the on-chip DRAM space shared cache mode. In this mode, the cache functions as a common cache for instructions and data for the on-chip DRAM, and all bus access of the DRAM memory space is cached. It is assumed that the microcomputer system uses the on-chip DRAM as the main memory.

Both the chip bus and the memory bus allow 128-bit parallel data transfer and caching is performed using direct mapping and store-in methods.

If the external memory space is accessed, data is transferred between the CPU and bus slave on the system bus via the bus interface unit (BIU). The BIU has a 128-bit data buffer, and converts between the chip bus and the system bus widths. Note that in this case, data being transferred is not cached.

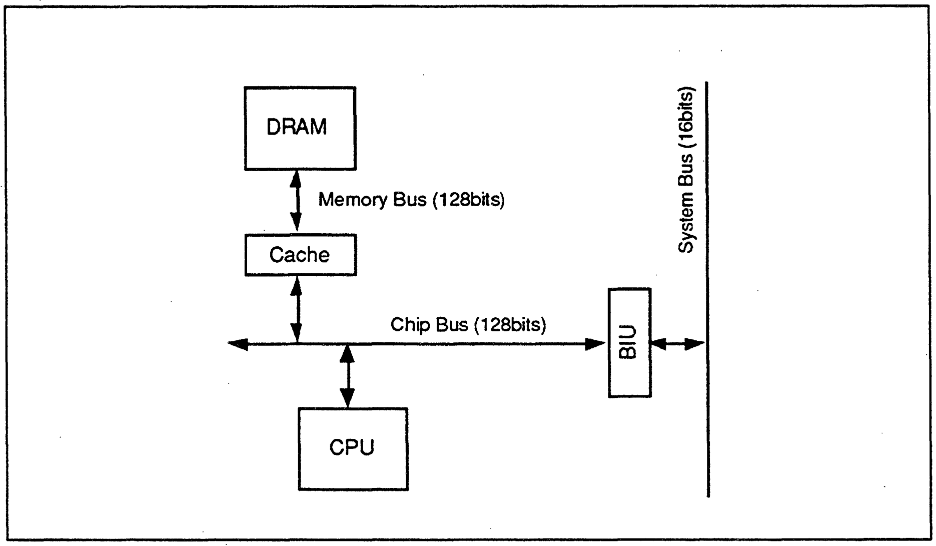


Figure 4.1 On-chip DRAM Space Shared Cache Mode

4.2.2 External User Space Instruction Cache Mode

Figure 4.2 shows the configuration of the M32R/DRAM internal memory system when selecting the external user space instruction cache mode. In this mode, the cache functions as an instruction cache for the external user space, and caches instruction fetch access of the system bus. It is assumed that the external ROM is used as a program memory and the internal DRAM is used as a data memory.

Caching is performed using direct mapping. Coherence with the instruction codes in the cache memory is not preserved when new instruction codes are written to the external user space.

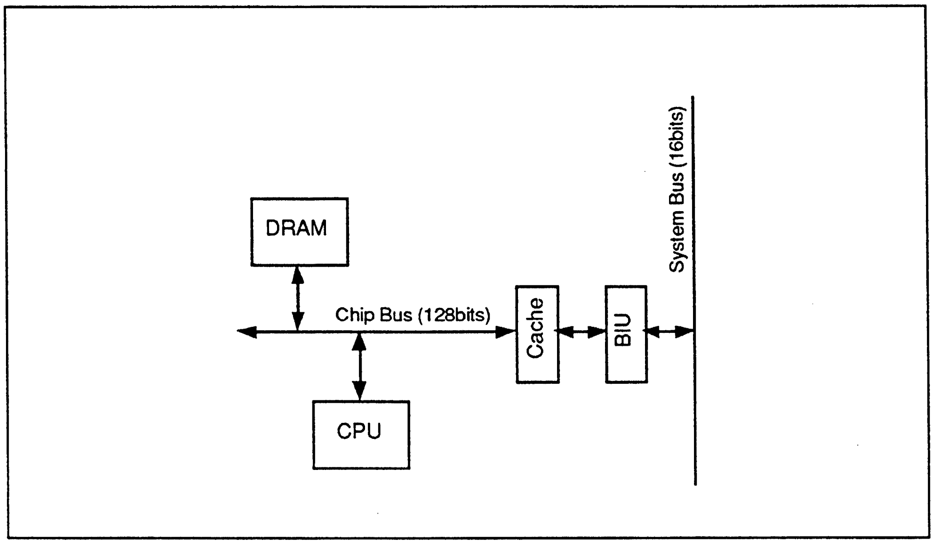


Figure 4.2 External User Space Instruction Cache Mode

4.2.3 Cache-off Mode

Figure 4.3 shows the configuration of the M32R/DRAM internal memory system when selecting the cache-off mode. In this mode, the cache is off and all bus cycles are directly applied to DRAM or to the system bus.

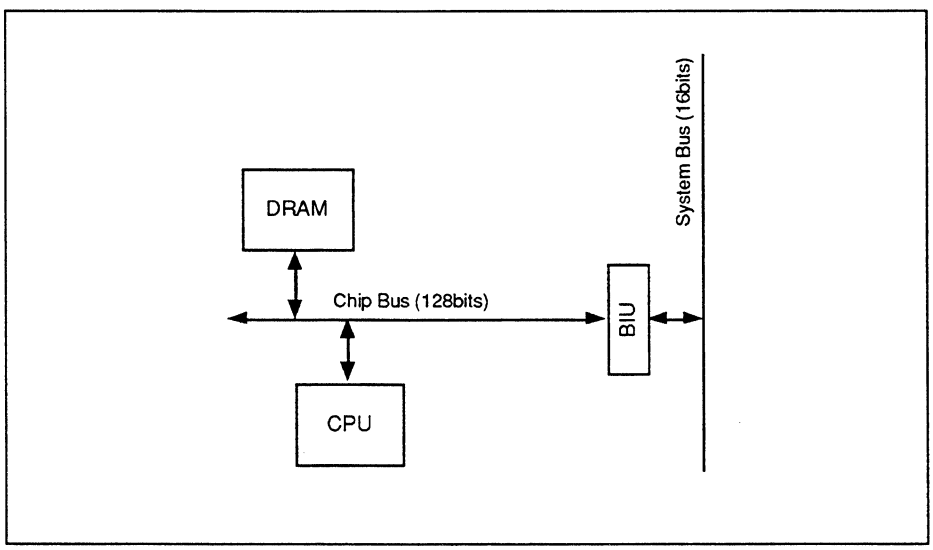


Figure 4.3 Cache-off Mode

5

Internal Peripheral I/O

5.1 Memory Controller

The memory controller is an internal peripheral I/O of the M32R and controls the on-chip DRAM and cache (SRAM). Its principal functions are as follows :

- Access and refresh of DRAM
- Setting the cache mode (i.e., sets configuration of the internal memory system)
- Controlling purge of the cache

5.1.1 Internal Memory System Control Registers

The M32R has the following three internal memory system control registers to define and control the internal memory system with the memory controller :

- Cache Purge Control Register (MSPR)
- DRAM Refresh Control Register (MDRR)
- Memory Controller Configuration Register (MCCR)

Figure 5.1 contains a memory map of these registers.

Logical Address	+0 address (b0 – b7)	+1 address (b8 – b15)	+2 address (b16 – b23)	+3 address (b24 – b31)
h'FFFFFFF4				MSPR
h'FFFFFFF8				MDRR
h'FFFFFFFC				MCCR

Figure 5.1 Memory Map of Internal Memory System Control Registers

5.1.1.1 Cache Purge Control Register (MSPR)

The cache purge control register (MSPR) specifies the purging of the on-chip SRAM cache. Writing a "1" to the purge specification bit purges the cache.

The bit structure of MSPR is shown in Table 5.1 :

Table 5.1 MSPR

b	Name	Function	Ini	R	W
0 | 6	(No mapping)		0	0	×
7	purge specification bit	0 : No purge. 1 : Purge.	0	0	o

- Purge Specification Bit (b7)

  0 : The cache is not purged.

  1 : The cache is purged after writing the data in the cache back to the DRAM.

(Note) Conventions

This chapter uses the following conventions in tables :

Term	Contents
b	Indicates the bit position (b0-b7)
Function	Shows the function that corresponds to each bit pattern × : indicates any value (0 or 1).
Ini	Shows bit value immediately after a reset has been cancelled. 0 : 0 (fixed) 1 : 1 (fixed) ? : unfixed
R	Shows read attribute o : read enabled 0 : fixed to 0 when reading × : read disabled 1 : fixed to 1 when reading
W	Shows write attribute o : write enabled × : write disabled (write requests ignored)

5.1.1.2 DRAM Refresh Control Register (MDRR)

The DRAM refresh control register specifies the on-chip DRAM refresh mode.

When the refresh mode bit is set to "0", the memory controller automatically refreshes the on-chip DRAM at regular intervals. (Auto refresh is equivalent to the CAS-before-RAS refresh used in standard DRAMs.)

The M32R on-chip DRAM has a self-refresh function that internally continues a low power dissipation refresh operation to enable the system to be backed up by battery. When the refresh mode bit of the DRAM refresh control register is set to "1", the memory controller starts the self-refresh cycle, and the DRAM stays in the self-refresh cycle as long as the register is set to "1". To cancel the self-refresh mode and return to the auto-refresh mode, the refresh mode select bit is set to "0".

Starting the self-refresh cycle sets the DRAM controller in standby (sleep) mode. During the self-refresh period, the memory controller does not accept requests to access the DRAM space. However, external space and internal I/O registers can be accessed at this time.

The bit structure of MDRR is shown in Table 5.2 :

Table 5.2 MDRR

b	Name	Function	Ini	R	W
0 | 6	(No mapping)		0	0	×
7	Refresh mode bit	0 : Auto-refresh mode 1 : Self-refresh mode	0	○	○

- Refresh Mode Bit (b7)
  The refresh mode bit sets refresh mode of the on-chip DRAM.

  0 : **auto-refresh mode**
  The memory controller performs a refresh at regular intervals.

  1 : **self-refresh mode**
  The memory controller the self-refresh mode

5.1.1.3 Memory Controller Configuration Register (MCCR)

The memory controller configuration register specifies the internal cache memory mode and the latency when accessing the on-chip DRAM. The bit structure is as shown in Table 5.3.

Table 5.3 MCCR

b	Name	Function	Ini	R	W
0 | 4	(No mapping)		0	0	×
5	Timing parameter bit	0 : Fast mode 1 : Slow mode	0	o	o
6 7	Cache mode bits	b6 b7 0 X : Cache-off mode 1 0 : On-chip DRAM space shared cache mode 1 1 : External user space instruction cache mode	0	o	o

- Timing Parameter Bit (b5)
  The memory controller optimizes the timing of the internal DRAM access according to the internal operating frequency.

  0 : Set to "0" when CLKIN is more than 8.3MHz.
  1 : Set to "1" when CLKIN is less than 8.3MHz.

||||| NOTE |||||

Note that correct operation cannot be guaranteed if a "1" is set when CLKIN is more than 8.3MHz.

- Cache Mode Bits (b6,b7)
  The cache mode bits specify the on-chip SRAM cache mode.

  0X : **cache-off mode**
  Inhibits cache operation

  10 : **on-chip DRAM space shared cache mode**

Both instructions and data are cached when the on-chip DRAM space is accessed.

11 : **external user space instruction cache mode**
Instruction fetches are cached when the external user space is accessed.

5.2 Programmable I/O Ports

The M32R has two programmable I/O ports (PP0 and PP1), each containing a data register and a direction control register.

5.2.1 Programmable Port Control Registers

The M32R has four control registers and four (programmable port direction control registers) for each programmable port. PPDR0 and PPCR0 are for PP0. PPDR1 and PPCR1 are for PP1.

- Programmable Port Data Registers (PPDR0, PPDR1)
- Programmable Port Direction Control Registers (PPCR0, PPCR1)

Figure 5.2 contains a memory map of these registers.

Logical Address	+0 address b0 – b7	+1 address b8 – b15	+2 address b16 – b23	+3 address b24 – b31
h'FFFFFFE0				PPCR0
h'FFFFFFE4				PPCR1
h'FFFFFFE8				PPDR0
h'FFFFFFEC				PPDR1

Figure 5.2 Memory Map of Programmable Port Control Registers

The software can read data from or write data to the programmable I/O ports. Each port can be individually controlled by its direction register for use as an input port or an output port.

5.2.1.1 Programmable Port Direction Control Registers (PPCR0, PPCR1)

The PPCR0 and PPCR1 registers control the programmable ports to specify input or output. PPCR0 controls port PP0. PPCR1 controls port PP1. When the direction bit is "0", the corresponding port is set to input; when "1", the port is set for output.

The bit structure of PPCR is shown in Table 5.4.

Table 5.4 PPCR0, PPCR1

b	Name	Function	Ini	R	W
0 | 6	(No mapping)		0	0	×
7	Port I/O direction specification bit	0 : Input mode 1 : Output mode	0	o	o

- Port I/O Direction Specification Bit (b7)

  0 : input mode

  1 : output mode

5.2.1.2 Programmable Port Data Registers (PPDR0, PPDR1)

When using the programmable I/O ports to output data externally or to input data from outside, set the output data to the data register, or read the data from the data register.

The ports are set for use as input ports or output ports using the direction control registers. (Refer to 5.2.1.1)

The bit structure of PPDR is as shown in Table 5.5.

Table 5.5 PPDR0,PPDR1

b	Name	Function	Ini	R	W
0 | 6	(No mapping)		0	o	×
7	Port data bit	0 : Data = "0" 1 : Data = "1"	?	o	o

- Port Data Bit (b7)

  0 : Port data = "0"
  1 : Port data = "1"

- When set for input mode

  The port pin is in the floating state, and the externally applied port pin level is stored in the port data register. Therefore, the input level of the port pin can be determined by reading the data bit corresponding to the input port.

  After a value has been written from outside to the data register in the input mode, the initial value of the output port can be set by switching to the output mode.

- When set for output mode

  When sending data via the port, the port direction control register is set to "1" after writing the value to the corresponding data bit. The value written to the data bit appears at the output port.

6

External Bus Interface

6.1 External Bus Interface Signals

The M32R has external bus-related signals described in this section which control operation of the external bus.

6.1.1 Address (A8-A30)

The M32R uses 24-bit addresses (bits A8 to A31) for addressing the 16MB memory space. The lowest A31, is not output from the external pin. In the write cycle, the valid write byte of the 16-bit data bus is output as $\overline{BCH}$ and $\overline{BCL}$. In the read cycle, the whole 16 bits of the data bus are read. Internally, however, only the data in the specified byte position is transferred.

The address pin is bi-directional. When accessing the on-chip DRAM from outside when the M32R is in hold state, the address is input from the system bus.

6.1.2 Space Identifier (SID)

In addition to the address pin, a space ID signal is output for discriminating between the user space and I/O space.

- User space SID=0
- I/O space SID=1

6.1.3 Byte Control ($\overline{BCH}$, $\overline{BCL}$)

The Byte Control signals show the byte position of the valid data during the external bus cycle. $\overline{BCH}$ corresponds to the high order byte and $\overline{BCL}$ to the low order byte. In the read bus cycle, the M32R sets both $\overline{\text{BCH}}$ and $\overline{\text{BCL}}$ to “L”. In the write bus cycle, $\overline{\text{BCH}}$ or $\overline{\text{BCL}}$ are set to “L” for the byte for which a write operation is requested.

When accessing the on-chip DRAM from outside, the byte control signal is input from the system bus.

6.1.4 Data Bus (D0-D15)

The data bus is a 16-bit data bus for external devices.

6.1.5 Bus Start ($\overline{\text{BS}}$)

When starting the bus cycle of the M32R to system bus, $\overline{\text{BS}}$ is set to “L” at the start of the bus cycle. For burst transfer, described later, $\overline{\text{BS}}$ is asserted for each transfer cycle.

The $\overline{\text{BS}}$ signal is not asserted when accessing internal resources such as the on-chip DRAM space or internal I/O registers.

6.1.6 Bus Status (ST)

The M32R outputs an ID signal to indicate if the bus cycle is for instruction read access or operand read/write access.

- Instruction fetch access ST=0
- Operand access ST=1
- Hold/idle ST= unspecified

6.1.7 Read/Write (R/$\overline{\text{W}}$)

The read/write (R/$\overline{\text{W}}$) signal is output to indicate if the external bus cycle is for read or write.

When accessing the on-chip DRAM from an external bus master, R/$\overline{\text{W}}$ is input from the system bus.

- Read bus cycle R/$\overline{\text{W}}$=1
- Write bus cycle R/$\overline{\text{W}}$=0

6.1.8 Burst (BURST)

For Burst the M32R starts two consecutive bus cycles which access 32-bit data aligned on 32-bit boundaries. In external instruction cache mode, the M32R starts eight consecutive bus cycles for reading 128-bit data aligned on 128-bit boundaries for cache data replacement. The M32R sets BURST to "L" during these consecutive bus cycles.

The address for 32-bit data access is output as the high order 16 bits followed by the low order 16 bits. The address for 128-bit data access is output starting at a 16-bit aligned address and continues in sequence for each access cycle so that it wraps around within a 128-bit boundary.

6.1.9 Data Complete (DC)

When the M32R starts the external bus cycle, the wait cycle is automatically inserted until DC is asserted from the bus slave on the system bus. Wait control using DC is also valid for bus cycles in a burst transfer.

The DC pin is bi-directional and if the on-chip DRAM is accessed by an external bus master while the M32R is in the hold state, the M32R asserts DC to inform the system bus that the bus cycle for the on-chip DRAM is completed.

6.1.10 Hold Request and Acknowledge (HREQ, HACK)

In the hold state, bus access by the M32R is stopped and all bus-related pins are in the high-impedance state. While the M32R is in the hold state, data transfer using the system bus can be done at any time.

HREQ "L" is asserted to set the M32R in the hold state. During the transition to the hold state, HACK is set to "L".

While in the hold state, the M32R pin status is shown in Figure 6.1. To return to the normal operating state from the hold state, HREQ is negated.

Table 6.1 Pin Status in Hold State

Pin	Pin Status and Operation
A8-30,D0-15,R/W,BCH,BCL,BURST	High Impedance
HACK	Output "L"
Others	Normal

6.1.11 On-chip DRAM Access Control (Chip Selector, $\overline{\text{CS}}$)

After the M32R is placed in the hold state, asserting to $\overline{\text{CS}}$ "L" enables the on-chip DRAM to be accessed from the system bus.

The following pins, which must be controlled from the system bus, are used to access the on-chip DRAM :

- A8-A30 — Specify the address in on-chip DRAM space using pins A8-A30 (On the memory map, A8-A9 should be "00", but even when not "00", the M32R accesses on-chip DRAM as if it were "00".)
- $\overline{\text{BCH}}$, $\overline{\text{BCL}}$ — Specify the byte control for the data to be written to the on-chip DRAM (When writing, only the byte for which $\overline{\text{BCH}}$ (high 16 bits) and $\overline{\text{BCL}}$ (low 16 bits) are asserted to "L" are written to DRAM. When reading, the halfword of data specified by A8-A30 is output to D0-D15 regardless of $\overline{\text{BCH}}$ and $\overline{\text{BCL}}$.)
- R/$\overline{\text{W}}$ — Specify read or write for the on-chip DRAM. ("H" corresponds to read, "L" to write.)
- D0-D15 — When writing to the on-chip DRAM, the write data is asserted. Specify byte control using $\overline{\text{BCH}}$ and $\overline{\text{BCL}}$, described earlier.

6.2 Read/Write Bus Operations

6.2.1 Read Cycles

Figure 6.1 shows the timing for a CPU read cycle with 0 wait states. Figure 6.2 shows the timing for a CPU read cycle with 1 wait. " o" indicates the sampling point.

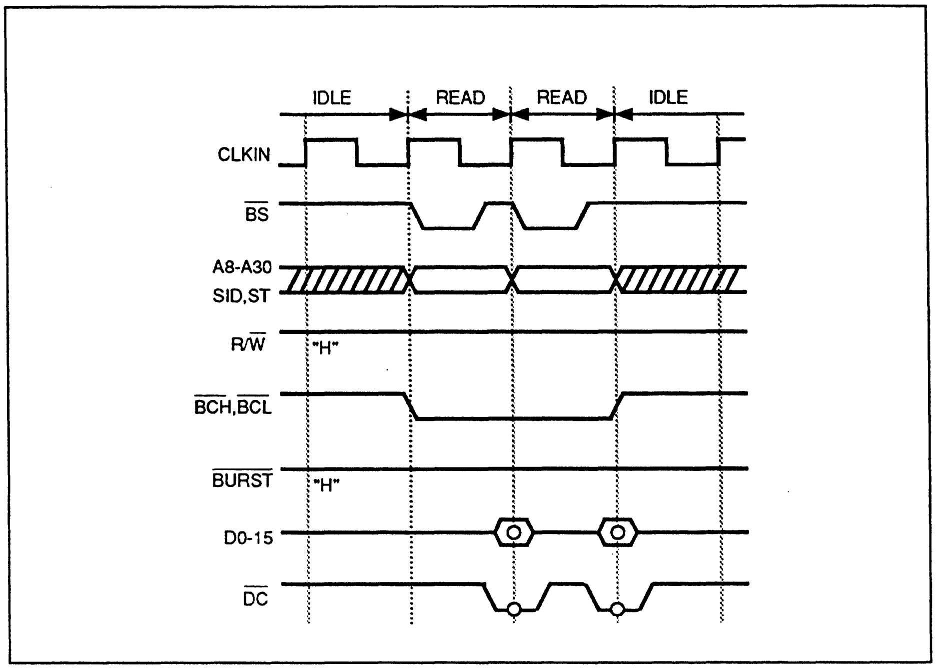


Figure 6.1 CPU Read Cycle (0 wait)

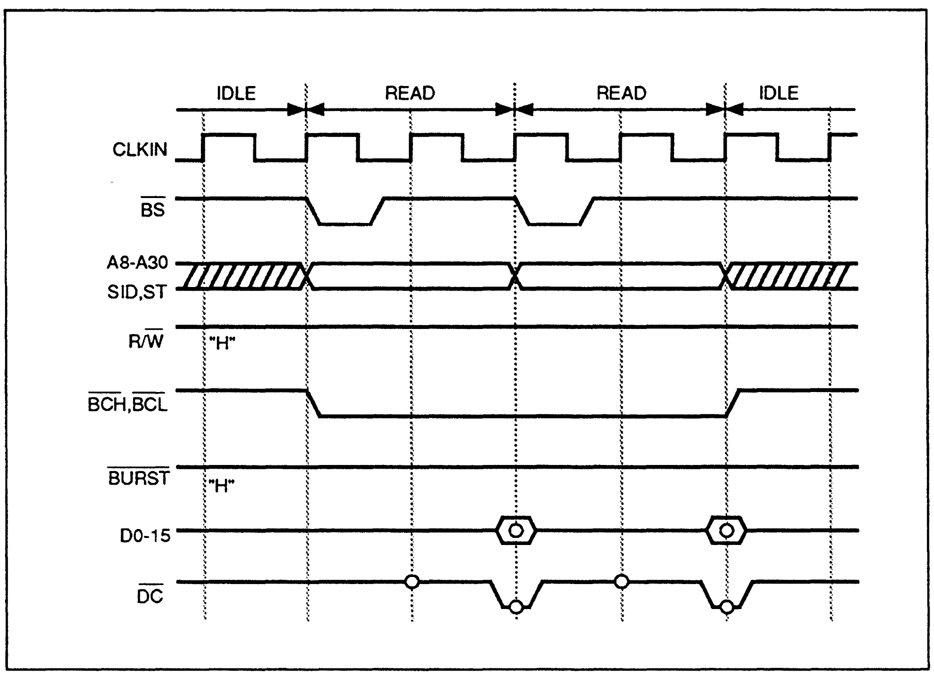


Figure 6.2 CPU Read Cycle (1 wait)

6.2.2 Burst Read Cycles

When reading word-size data aligned on a 32-bit boundary, or 4-word-size data aligned on a 128-bit boundary for cache replacement in the external instruction cache mode, assert BURST. Figures 6.3 and 6.4 show the read bus cycle in each of these cases.

When reading 32-bit data in burst, always start the bus cycles with the 16-bit bus read cycle for the high order (MSB) followed by that for the low order (LSB). When burst reading 128-bit data, start the bus cycles with any address on a 16-bit boundary, then 8 cycles for the 8-to-16-bit parts of the 128 bits in sequence so that all data on the 128 bit line is read. Address bits A8 to A30 are output each cycle.

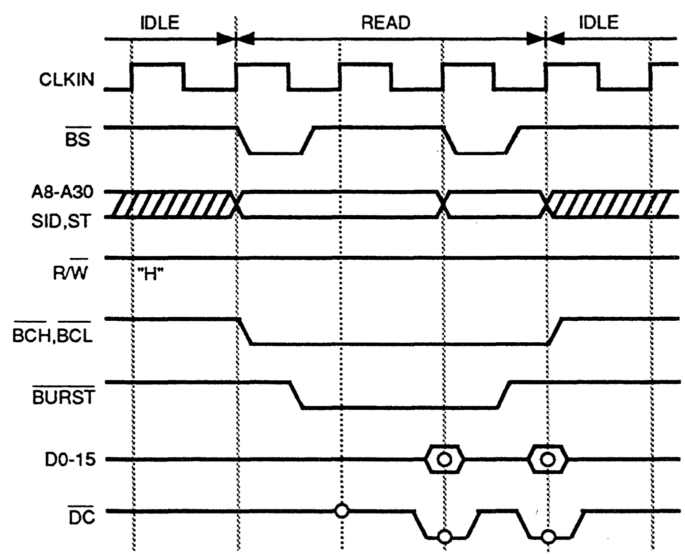


Figure 6.3 CPU Burst Read Cycle
(When reading word-size data aligned on a 32-bit boundary)

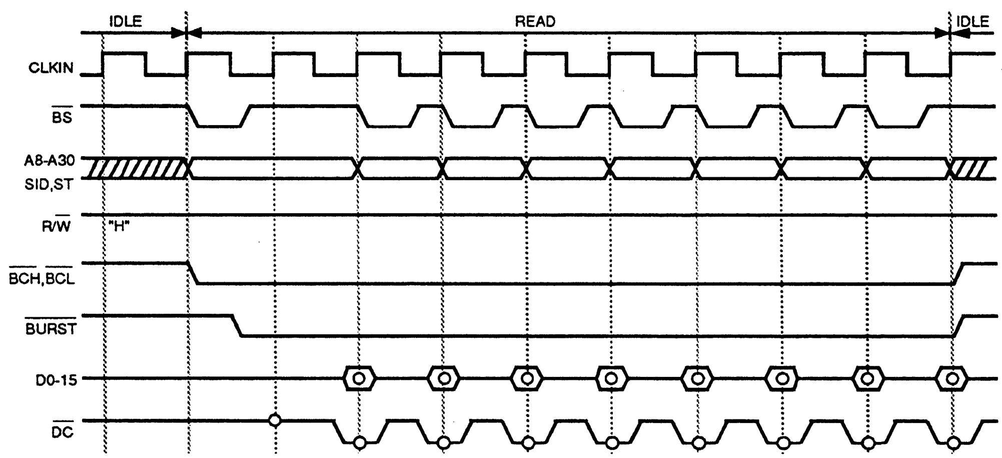


Figure 6.4 CPU Burst Read Cycle
(4-word-size data aligned on a 128-bit boundary)

6.2.3 Write Cycles

Figure 6.5 shows the timing for a CPU write cycle with 0 wait states. Figure 6.6 shows the timing for a CPU write cycle with 1 wait state. " o" indicates the sampling point.

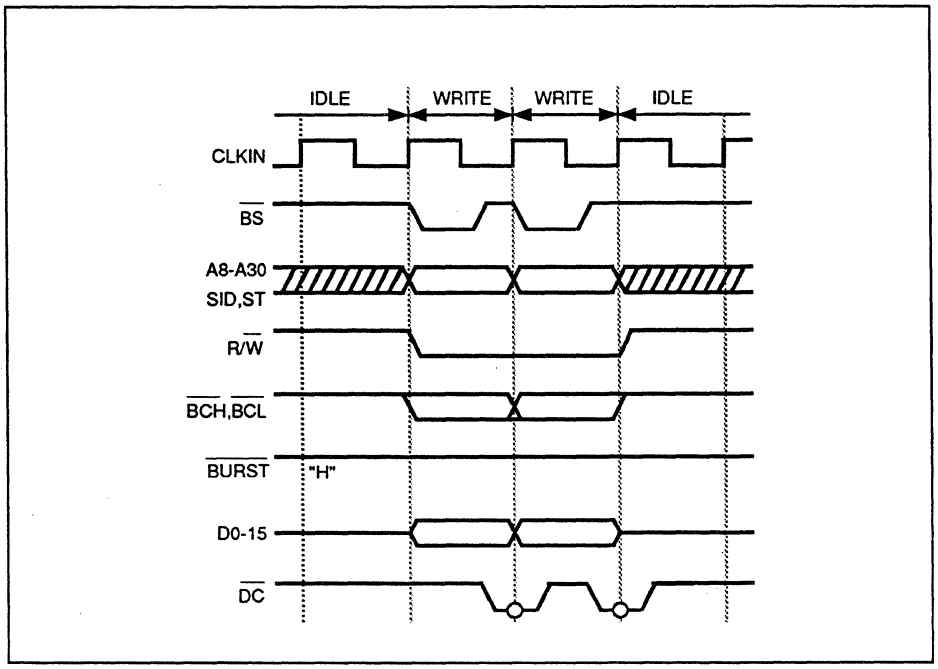


Figure 6.5 CPU Write Cycle (0 wait)

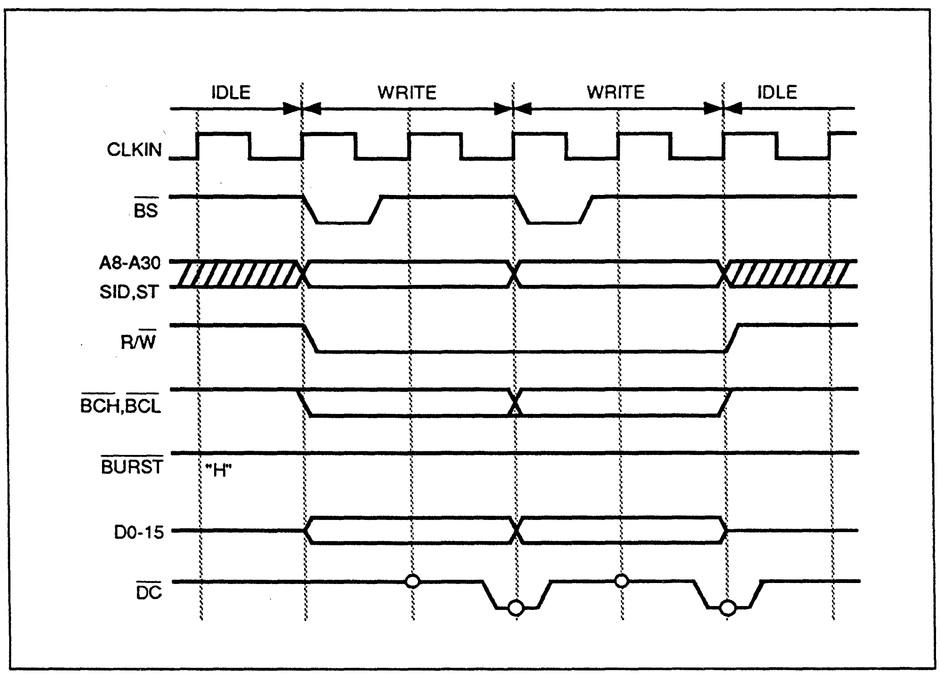


Figure 6.6 CPU Write Cycle (1 wait)

6.2.4 Burst Write Cycles

When writing word-size data aligned on a 32-bit boundary, assert $\overline{BURST}$. Figure 6.7 shows the bus cycle.

When burst writing 32-bit data, always start the bus cycles with the 16-bit bus write cycle for the high order (MSB) followed by that for the low order (LSB). Addresses A8 to A30 are output each cycle.

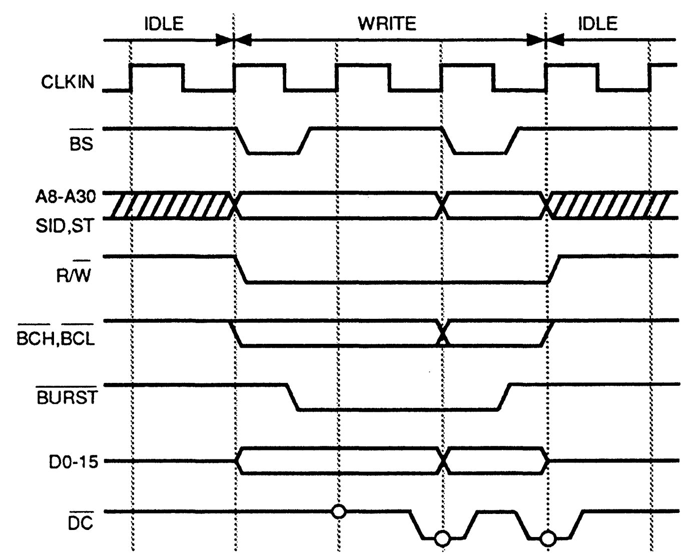


Figure 6.7 CPU Burst Write Cycle

6.2.5 Continuous Bus Cycle

As shown in Figure 6.8, the M32R inserts an idle cycle immediately after the read bus cycle (without starting the write bus cycle) to avoid data collision on the system bus. This also applies to the write cycle (burst write cycle) immediately after the burst read.

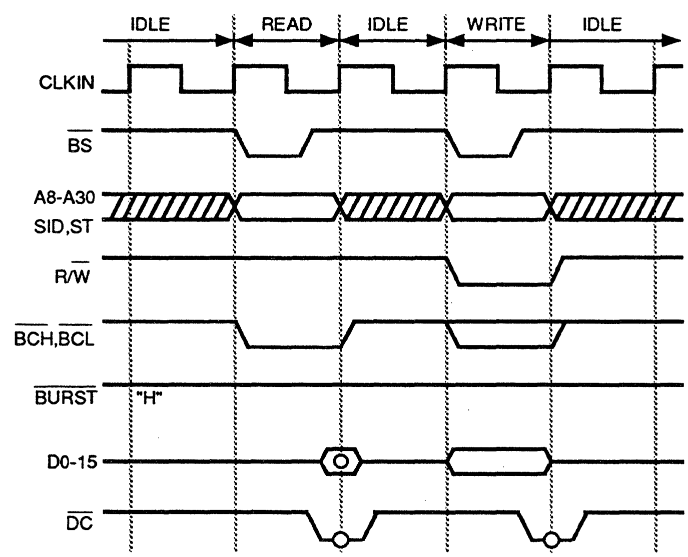


Figure 6.8 Write Cycle Immediately After Read Cycle

6.3 Hold Cycles

When $\overline{HREQ}$ is set to "L", $\overline{HACK}$ is set to "L" within several clocks and the M32R enters a hold state. As shown in Figure 6.9, bus-related pins are set to high-impedance while the M32R is in the hold state so that data transfers can be performed when required on the system bus.

Negate $\overline{HREQ}$ to return from the hold state to normal operation.

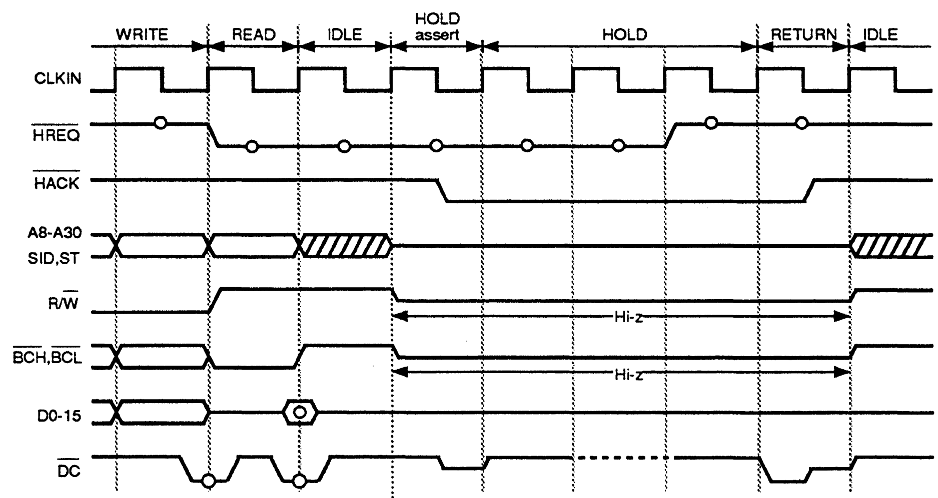


Figure 6.9 Transition to and from Hold State

6.4 External Bus Master Access

After setting the M32R to the hold state, setting $\overline{CS}$ to "L" enables the on-chip DRAM to be accessed from the system bus.

6.4.1 External Bus Master Read Cycles

As shown in Figure 6.10, if $\overline{CS}$ is asserted while the M32R is in the hold state, it is interpreted as a request for bus access of the on-chip DRAM and the internal memory controller starts the bus cycle for DRAM. When the R/$\overline{W}$ pin is "H", a read bus cycle is assumed. On completion of the bus cycle, the read data from DRAM is output on D0-D15 and $\overline{DC}$ is asserted. In the case of a read bus cycle, the 16 bits of data at the address specified by A8-A30 is output regardless of what is specified by $\overline{BCH}$ and $\overline{BCL}$.

In the M32R, the 128 bits of data including the requested address are temporarily read into the 128-bit data buffer in the bus interface unit (BIU). Therefore, when reading contiguous addresses within a 128-bit boundary, the next read bus cycle does not include the read cycle from DRAM and therefore ends within 1 ×CLKIN. Two or three CLKIN are required for the first bus access.

When the target read bus cycle has finished, immediately negate $\overline{CS}$. Also, when negating $\overline{HREQ}$ to return from the hold state to normal operation, do so at the same time or after negating $\overline{CS}$.

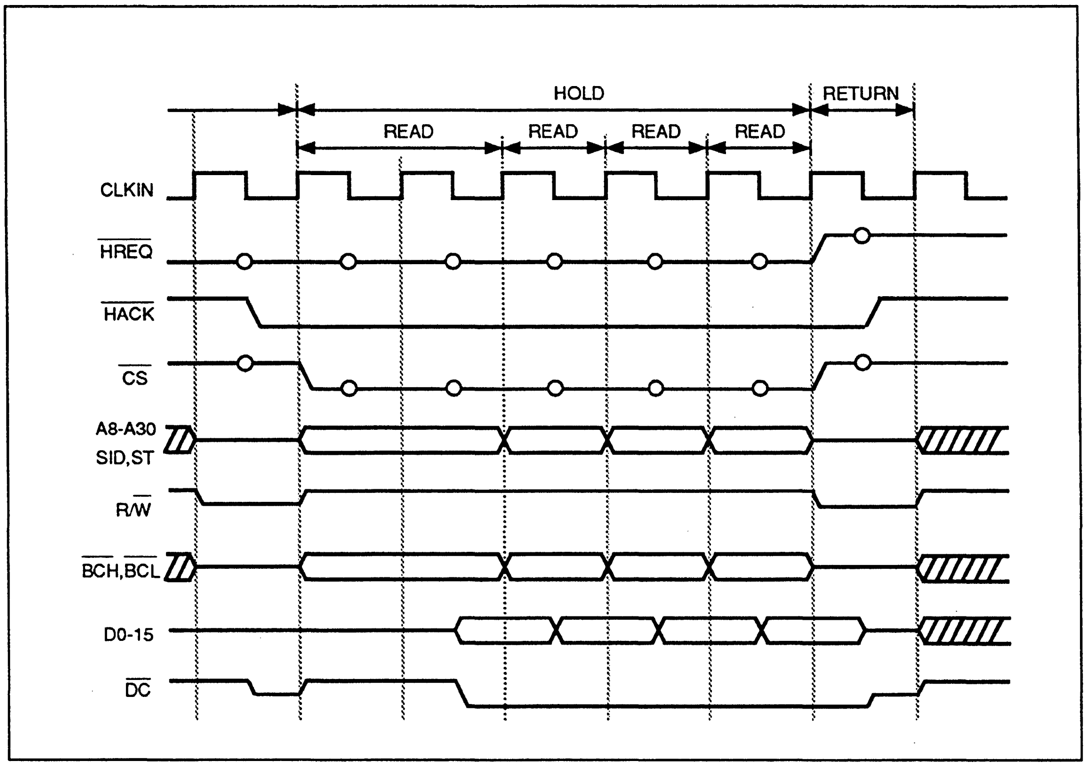


Figure 6.10 Read Bus Cycle of On-chip DRAM

6.4.2 External Bus Master Write Cycles

As shown in Figure 6.11, if CS is asserted while the M32R is in the hold state, the M32R interprets this as a request for bus access of on-chip DRAM and the internal memory controller starts the bus cycle for the DRAM. On completion of the bus cycle, DC is asserted. The byte data control is specified using the BCH and BCL pins. Only the data in the byte positions for which BCH and BCL are asserted "L" is written to DRAM.

In the M32R, the write-requested data is temporarily stored in the 128-bit data buffer in the bus interface unit (BIU) before being written to DRAM. This reduces the frequency of DRAM access and increases the throughput when data is to be written to contiguous addresses. Therefore, continuous write accesses within a 128-bit boundary are completed within 1 ×CLKIN. Figure 6.11 shows an example of four bus access in which the first and second are to continuous addresses within one 128-bit boundary and the third and fourth are to continuous addresses within a second 128-bit boundary.

If the target write bus cycle has finished, immediately negate CS. Also, if negate

HREQ is negated to return from the hold state to normal operation, it most be the same time or after the negating of CS.

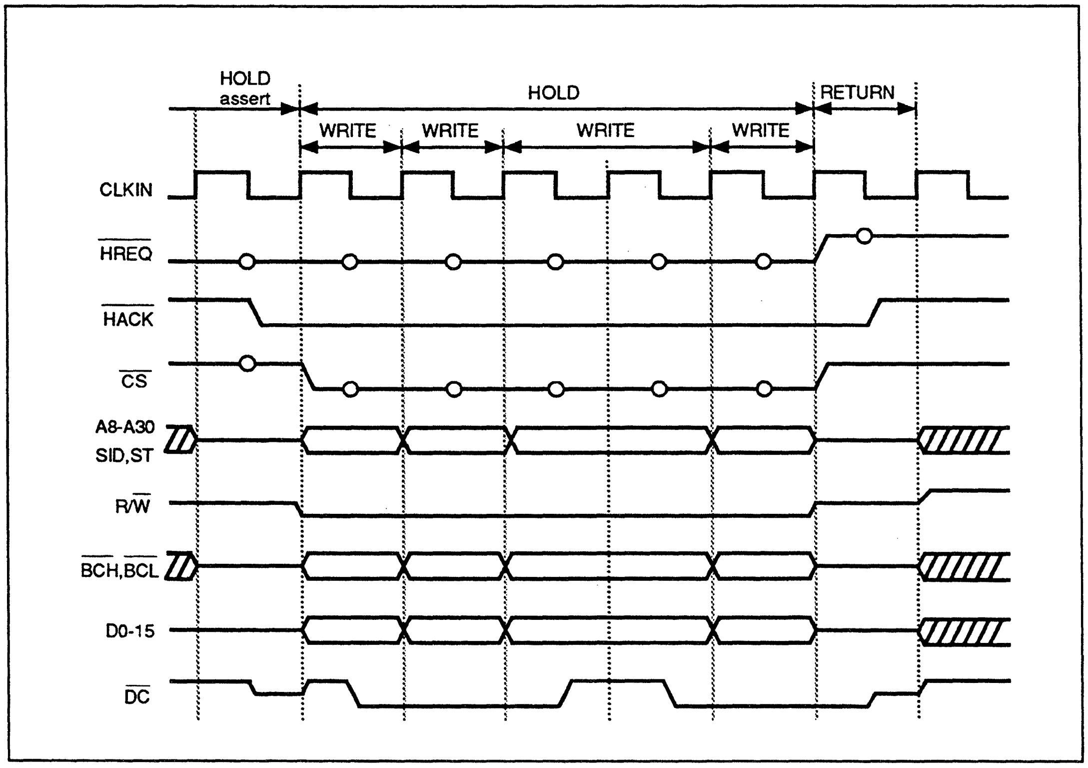


Figure 6.11 Write Bus Cycle of On-chip DRAM

6.5 Reset

Asserting $\overline{RST}$ the "L" resets the M32R. The reset state is cancelled by negating $\overline{RST}$, and program execution starts at the address shown by the reset vector.

6.5.1 Reset Operation

When the power is on, $\overline{RST}$ can be asserted to "L" to start the ×4 clock generator in the M32R. Keep $\overline{RST}$ asserted for at least 2 ms after VCC has stabilized at the specified voltage level.

To reset the M32R during operation, assert $\overline{RST}$ for a minimum of 4 ×CLKIN cycles.

6.5.2 Signal States During Reset

While $\overline{RST}$ is asserted and immediately after $\overline{RST}$ is negated, the external pins of the M32R are in the following states :

- While $\overline{RST}$ is asserted and for 5×CLKIN after $\overline{RST}$ is negated
  →Undefined
- For 5 to 8×CLKIN after $\overline{RST}$ is negated
  →Idle
- At 15×CLKIN after $\overline{RST}$ is negated
  →The vector fetch bus cycle starts

6.5.3 CPU Internal States After Reset Sequence

When an active $\overline{RST}$ signal goes "H", the M32R CPU registers are initialized as shown in Table 6.2.

Table 6.2 Contents of Control Registers After Reset Sequence

Register		Content
PSW (CR0)		B'0000 0000 0000 0000 ??00 000? 0000 0000 (BSM, BIE, and BC are undefined)
CBR (CR1)		H'0000 0000
SPI (CR2)		H'0000 0000
SPU (CR3)		H'0000 0000
BPC (CR6)		undefined
PC (program counter)		H'7FFF FFF0 (restart address)
ACC (accumulator)		undefined
Programmable I/O	PPCR0, PPCR1	H'00 (input mode)
	PPDR0, PPDR1	B'0000 000? (port data are dependent on the pin status)
Memory Control	MSPR	H'00 (no purge)
	MDPR	H'00 (auto-refresh mode)
	MCCR	H'00 (fast mode, cache-off mode)

7

Electrical Characteristics

7.1 Absolute Maximum Ratings

Table 7.1 Absolute Maximum Ratings

Symbol	Parameter	Conditions	Ratings		Unit
			Min.	max.	
VCC	Supply voltage		-0.5	4.6	V
VI	Input voltage		-0.5	4.6	V
VO	Output voltage		-0.5	4.6	V
PD	Power dissipation	Ta = 25°C		TBD	mW
ICC	VCC power supply current			TBD	mA
Topr	Operating temprature		0	70	°C
Tstg	Storege temprature		-65	150	°C

7.2 Recommended Operating Conditions

Table 7.2 Recommended Operating Conditions

Symbol	Parameter	Linits			Unit
		Min.	Typ.	Max.	
VCC	Supply voltage	3.0	3.3	3.6	V
VSS	Supply voltage	0	0	0	V
VIH	High level input voltage	2.0		VCC+0.3	V
VIL	Low level input voltage	-0.3		0.8	V
CL	Output load capacitance			50	pF

(Ta=0 to 70°C unless otherwise specified)

7.3 M32R DC/AC Characteristics

T.B.D

Appendix A:

The M32R Instruction Set

A.1 List of Instructions

Mnemonic	Operand(s)	Function	*C(condition bit)*
ADD	Rdest,Rsrc	Rdest = Rdest + Rsrc	
ADD3	Rdest,Rsrc,#c16	Rdest = Rsrc + (sh)c16	
ADDI	Rdest,#c8	Rdest = Rdest + (sb)c8	
ADDV	Rdest,Rsrc	Rdest = Rdest + Rsrc	*overflow*
ADDV3	Rdest,Rsrc,#c16	Rdest = Rsrc + (sh)c16	*overflow*
ADDX	Rdest,Rsrc	Rdest = Rdest + Rsrc + C	*carry*
AND	Rdest,Rsrc	Rdest = Rdest & Rsrc	
AND3	Rdest,Rsrc,#c16	Rdest = Rsrc & (uh)c16	
BC	c8	if(C) PC=PC+((sb)c8<<2)	
BC	c24	if(C) PC=PC+((s24)c24<<2)	
BEQ	Rdest,Rsrc,c16	if(Rsrc == Rdest) PC=PC+((sh)c16<<2)	
BEQZ	Rsrc,c16	if(Rsrc == 0) PC=PC+((sh)c16<<2)	
BGEZ	Rsrc,c16	if(Rsrc >= 0) PC=PC+((sh)c16<<2)	
BGTZ	Rsrc,c16	if(Rsrc > 0) PC=PC+((sh)c16<<2)	
BL	c8	R14=PC+4,PC=PC+((sb)c8<<2)	
BL	c24	R14=PC+4,PC=PC+((s24)c24<<20	
BLEZ	Rsrc,c16	if(Rsrc <= 0) PC=PC+((sh)c16<<2)	
BLTZ	Rsrc,c16	if(Rsrc < 0) PC=PC+((sh)c16<<2)	
BNC	c8	if(!C) PC=PC+((sb)c8<<2)	
BNC	c24	if(!C) PC=PC+((s24)c24<<20	
BNE	Rdest,Rsrc,c16	if(Rsrc != Rdest) PC=PC+((sh)c16<<2)	
BNEZ	Rsrc,c16	if(Rsrc != 0) PC=PC+((sh)c16<<2)	
BRA	c8	PC=PC+((sb)c8<<2)	
BRA	c24	PC=PC+((s24)c24<<2)	
CMP	Rsrc1,Rsrc2	(s)Rsrc1 < (s)Rsrc2	
CMPI	Rsrc,#c16	(s)Rsrc < (sh)c16	
CMPU	Rsrc1,Rsrc2	(u)Rsrc1 < (u)Rsrc2	

CMPUI	Rsrc,#c16	(u)Rsrc < (u)((sh)c16)
DIV	Rdest,Rsrc	Rdest = (s)Rdest / (s)Rsrc
DIVU	Rdest,Rsrc	Rdest = (u)Rdest / (u)Rsrc
JL	Rsrc	R14 = PC+4, PC = Rsrc
JMP	Rsrc	PC = Rsrc
LD	Rdest,@(c16,Rsrc)	Rdest = *(s *)(Rsrc+(sh)c16)
LD	Rdest,@Rsrc	Rdest = *(s *)Rsrc
LD	Rdest,@Rsrc+	Rdest = *(s *)Rsrc, Rsrc += 4
LD24	Rdest,#c24	Rdest = c24 & 0x00ffffff
LDB	Rdest,@(c16,Rsrc)	Rdest = *(sb *)(Rsrc+(sh)c16)
LDB	Rdest,@Rsrc	Rdest = *(sb *)Rsrc
LDH	Rdest,@(c16,Rsrc)	Rdest = *(sh *)(Rsrc+(sh)c16)
LDH	Rdest,@Rsrc	Rdest = *(sh *)Rsrc
LDI	Rdest,#c16	Rdest = (sh)c16
LDI	Rdest,#c8	Rdest = (sb)c8
LDUB	Rdest,@(c16,Rsrc)	Rdest = *(ub *)(Rsrc+(sh)c16)
LDUB	Rdest,@Rsrc	Rdest = *(ub *)Rsrc
LDUH	Rdest,@(c16,Rsrc)	Rdest = *(uh *)(Rsrc+(sh)c16)
LDUH	Rdest,@Rsrc	Rdest = *(ub *)Rsrc
LOCK	Rdest,@Rsrc	LOCK = 1, Rdest = *(s *)Rsrc
MACHI	Rsrc1,Rsrc2	accumulator += (s)(Rsrc1 & 0xffff0000) * (s)((s)Rsrc2>>16)
MACLO	Rsrc1,Rsrc2	accumulator += (s)(Rsrc1<<16) * (sh)Rsrc2
MACWHI	Rsrc1,Rsrc2	accumulator += (s)Rsrc1 * (s)((s)Rsrc2>>16)
MACWLO	Rsrc1,Rsrc2	accumulator += (s)Rsrc1 * (sh)Rsrc2
MUL	Rdest,Rsrc	Rdest = (s)Rdest * (s)Rsrc
MULHI	Rsrc1,Rsrc2	accumulator = (s)(Rsrc1 & 0xffff0000) * (s)((s)Rsrc2>>16)
MULLO	Rsrc1,Rsrc2	accumulator = (s)(Rsrc1<<16) * (sh)Rsrc2
MULWHI	Rsrc1,Rsrc2	accumulator = (s)Rsrc1 * (s)((s)Rsrc2>>16)
MULWLO	Rsrc1,Rsrc2	accumulator = (s)Rsrc1 * (sh)Rsrc2
MV	Rdest,Rsrc	Rdest = Rsrc
MVFACHI	Rdest	Rdest = accumulater >> 32
MVFACLO	Rdest	Rdest = accumulator >> 16
MVFACMI	Rdest	Rdest = accumulator
MVFC	Rdest,CRsrc	Rdest = CRsrc
MVTACHI	Rsrc	accumulator[0:31] = Rsrc
MVTACLO	Rsrc	accumulator[32:63] = Rsrc
MMVTC	Rsrc,CRdest	CRdest = Rsrc
NEG	Rdest,Rsrc	Rdest = 0 - Rsrc
NOP	/*no-operation*/	
NOT	Rdest,Rsrc	Rdest = ~Rsrc

OR	Rdest,Rsrc	Rdest = Rdest | Rsrc	
OR3	Rdest,Rsrc,#c16	Rdest = Rsrc | (uh)c16	
RAC		Round the 16-bit value in the accumulator	
RACH		Round the 32-bit value in the accumulator	
REM	Rdest,Rsrc	Rdest = (s)Rdest % (s)Rsrc	
REMU	Rdest,Rsrc	Rdest = (u)Rdest % (u)Rsrc	
SETH	Rdest,#c16	Rdest = c16 << 16	
RTE		PC = BPC & 0xfffffffc, PSW[SM,IE,C] = PSW[BSM,BIE,BC]	
SLL	Rdest,Rsrc	Rdest = Rdest << (Rsrc & 31)	
SLL3	Rdest,Rsrc,#c16	Rdest = Rsrc << (c16 & 31)	
SLLI	Rdest,#c5	Rdest = Rdest << c5	
SRA	Rdest,Rsrc	Rdest = (s)Rdest >> (Rsrc & 31)	
SRA3	Rdest,Rsrc,#c16	Rdest = (s)Rsrc >> (c16 & 31)	
SRAI	Rdest,#c5	Rdest = (s)Rdest >> c5	
SRL	Rdest,Rsrc	Rdest = (u)Rdest >> (Rsrc & 31)	
SRL3	Rdest,Rsrc,#c16	Rdest = (u)Rsrc >> (c16 & 31)	
SRLI	Rdest,#c5	Rdest = (u)Rdest >> c5	
ST	Rsrc1,@(c16,Rsrc2)	*(s *)(Rsrc2+(sh)c16) = Rsrc1	
ST	Rsrc1,@+Rsrc2	Rsrc2 += 4, *(s *)Rsrc2 = Rsrc1	
ST	Rsrc1,@-Rsrc2	Rsrc2 -= 4, *(s *)Rsrc2 = Rsrc1	
ST	Rsrc1,@Rsrc2	*(s *)Rsrc2 = Rsrc1	
STB	Rsrc1,@(c16,Rsrc2)	*(sb *)(Rsrc2+(sh)c16) = Rsrc1	
STB	Rsrc1,@Rsrc2	*(sb *)Rsrc2 = Rsrc1	
STH	Rsrc1,@(c16,Rsrc2)	*(sh *)(Rsrc2+(sh)c16) = Rsrc1	
STH	Rsrc1,@Rsrc2	*(sh *)Rsrc2 = Rsrc1	
SUB	Rdest,Rsrc	Rdest = Rdest - Rsrc	
SUBV	Rdest,Rsrc	Rdest = Rdest - Rsrc	*overflow*
SUBX	Rdest,Rsrc	Rdest = Rdest - Rsrc - C	*borrow*
TRAP	#n	trap	
UNLOCK	Rsrc1,@Rsrc2	if(LOCK) { *(s *)Rsrc2 = Rsrc1; } LOCK=0	
XOR	Rdest,Rsrc	Rdest = Rdest ^ Rsrc	
XOR3	Rdest,Rsrc,#c16	Rdest = Rsrc ^ (uh)c16	

where:

```
typedef singed int      s;    /* 32 bit signed integer (word)*/
typedef unsigned int    u;    /* 32 bit unsigned integer (word)*/
typedef signed short    sh;   /* 16 bit signed integer (halfword)*/
typedef unsigned short  uh;   /* 16 bit unsigned integer (halfword)*/
typedef signed char     sb;   /*  8 bit signed integer (byte)*/
typedef unsigned char   ub;   /*  8 bit unsigned integer (byte)*/
```

A.2 Detail Description of Instructions

For this section, refer to Chapter 2, "Programming model", for more details about registers, data formats, addressing modes, instruction formats, and the instruction set. The following details are given for each instruction :

Instruction	Shows the mnemonic and possible operands using assembly notation (Refer to section 2.3, "Addressing modes", for addressing modes and its notation.)
Function	Indicates the operation performed by the instruction (Notations are according to the C language notation.)
Description	Describes the operation performed and the condition bit change by the instruction
Exceptions	Shows the possible exceptions or traps
Encoding	Shows the instruction format(s) (i.e., the object code produced for the instruction)

ADD

Instruction

`ADD Rdest,Rsrc`

Function

`Rdest = Rdest + Rsrc`

Description

ADD adds Rsrc to Rdest and puts the result in Rdest.

The condition bit C is unchanged.

Exceptions

None

Encoding

0000	dest	1010	src

`ADD Rdest,Rsrc`

ADD3

Instruction

```
ADD3 Rdest,Rsrc,#imm16
```

Function

```
Rdest = Rsrc + (signed short)imm16
```

Description

ADD3 adds the 16-bit immediate value to Rsrc and puts the result in Rdest (The immediate value is sign-extended before it is added to Rsrc)

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1000	dest	1010	src	imm16	ADD3 Rdest,Rsrc,#imm16

ADDI

Instruction

```
ADDI Rdest,#imm8
```

Function

```
Rdest = Rdest + (signed char)imm8
```

Description

ADDI adds the 8-bit immediate value to Rdest and puts the result in Rdest (The immediate value is sign-extended before it is added to Rdest)

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0100	dest	imm8

ADDI Rdest,#imm8

ADDV

Instruction

```
ADDV Rdest,Rsrc
```

Function

```
Rdest = (signed)Rdest + (signed)Rsrc
C = overflow ? 1:0
```

Description

ADDV adds Rsrc to Rdest and puts the result in Rdest.

The condition bit, C, is set when the addition results in a two's-complement overflow; otherwise it is cleared.

Exceptions

None

Encoding

0000	dest	1000	src	ADDV Rdest,Rsrc

ADDV3

Instruction

```
ADDV3 Rdest,Rsrc,#imm16
```

Function

```
Rdest = (signed)Rsrc + (signed)((signed short)imm16)
C = overflow ? 1:0
```

Description

ADDV3 adds the 16-bit immediate value to Rsrc and puts the result in Rdest (The immediate value is sign-extended before it is added to Rsrc)

The condition bit, C, is set when the addition results in a two's-complement overflow; otherwise it is cleared.

Exceptions

None

Encoding

1000	dest	1000	src	imm16	ADDV3 Rdest,Rsrc,#imm16

ADDX

Instruction

```
ADDX Rdest,Rsrc
```

Function

```
Rdest = (unsigned)Rdest + (unsigned)Rsrc + C
C = carry-out ? 1:0
```

Description

ADDX adds Rsrc and C to Rdest and puts the result in Rdest

The condition bit, C, is set when the addition result can't be represented by a 32-bit unsigned integer; otherwise it is cleared.

Exceptions

None

Encoding

0000	dest	1001	src	ADDX Rdest,Rsrc

AND

Instruction

`AND Rdest,Rsrc`

Function

`Rdest = Rdest & Rsrc`

Description

AND computes the logical AND of the corresponding bits of Rdest and Rsrc and puts the result in Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0000	dest	1100	src	AND Rdest,Rsrc

AND3

Instruction

```
AND3 Rdest,Rsrc,#imm16
```

Function

```
Rdest = Rsrc & (unsigned short)imm16
```

Description

AND3 computes the logical AND of the corresponding bits of Rsrc and the 16-bit immediate value, with zero-extended from 16-bits to 32-bits and puts the result in Rdest.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1000	dest	1100	src	imm16	AND3 Rdest,Rsrc,#imm16

BC

Instruction

```
BC Label
(1) BC disp8
(2) BC disp24
```

Function

```
(1) if (C==1) PC = (PC & 0xfffffffc)+(((signed char)disp8)<< 2)
(2) if (C==1) PC = (PC & 0xfffffffc)+(sign_extend(disp24)<< 2)
```

where :

```
#define sign_extend(x) (((signed)((x)<< 8))>> 8)
```

Description

BC branches to the specified label when the condition bit C is 1

There are two instruction formats; which allows software, such as an assembler, the decision on the better format.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0111	1100	disp8	BC disp8

1111	1100	disp24	BC disp24

BEQ

Instruction

```
BEQ Rsrc1,Rsrc2,Label
```

Function

```
if (Rsrc1==Rsrc2) PC = (PC & 0xfffffffc)+(((signed short)disp16)<< 2)
```

Description

BEQ branches to the specified label when Rsrc1 is equal to Rsrc2.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1011	src1	0000	src2	disp16	BEQ Rsrc1,Rsrc2,disp16

BEQZ

Instruction

```
BEQZ Rsrc,Label
```

Function

```
if (Rsrc==0) PC = (PC & 0xfffffffc)+(((signed short)disp16)<< 2)
```

Description

BEQZ branches to the specified label when Rsrc is equal to zero

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1011	0000	1000	src	disp16	BEQZ Rsrc,disp16

BGEZ

Instruction

```
BGEZ Rsrc,Label
```

Function

```
if ((signed)Rsrc>=0)
    PC = (PC & 0xfffffffc)+(((signed short)disp16)<< 2)
```

Description

BGEZ branches to the specified label when the contents of Rsrc, treated as a signed 32-bit value, are greater than or equal to zero

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1011	0000	1011	src	disp16	BGEZ Rsrc,disp16

BGTZ

Instruction

```
BGTZ Rsrc,Label
```

Function

```
if ((signed)Rsrc>0)
    PC = (PC & 0xfffffffc)+(((signed short)disp16)<< 2)
```

Description

BGTZ branches to the specified label when the contents of Rsrc, treated as a signed 32-bit value, are greater than zero

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1011	0000	1101	src	disp16	BGTZ Rsrc,disp16

BL

Instruction

```
BL Label
(1) BL disp8
(2) BL disp24
```

Function

```
(1) R14 = (PC & 0xfffffffc)+4
    PC = (PC & 0xfffffffc)+(((signed char)disp8)<< 2)
(2) R14 = (PC & 0xfffffffc)+4
    PC = (PC & 0xfffffffc)+(sign_extend(disp24)<< 2)
```

where :

```
#define sign_extend(x)  (((signed)((x)<< 8))>> 8)
```

Description

BL branches unconditionally to the address specified by the label and puts the return address in R14

There are two instruction formats; which allows software, such as an assembler, the decision on the better format.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0111	1110	disp8	BL disp8

1111	1110	disp24	BL disp24

BLEZ

Instruction

```
BLEZ Rsrc,Label
```

Function

```
if ((signed)Rsrc<=0)
    PC = (PC & 0xfffffffc)+(((signed short)disp16)<< 2)
```

Description

BLEZ branches to the specified label when the contents of Rsrc, treated as a signed 32-bit value, are less than or equal to zero

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1011	0000	1100	src	disp16	
					BLEZ Rsrc,disp16

BLTZ

Instruction

```
BLTZ Rsrc,Label
```

Function

```
if ((signed)Rsrc<0)
    PC = (PC & 0xfffffffc)+(((signed short)disp16)<< 2)
```

Description

BLTZ branches to the specified label when the contents of Rsrc, treated as a signed 32-bit value, are less than zero

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1011	0000	1010	src	disp16	BLTZ Rsrc,disp16

BNC

Instruction

`BNC Label`

(1) `BNC disp8`

(2) `BNC disp24`

Function

(1) `if (C==0) PC = (PC & 0xfffffffc)+(((signed char)disp8)<< 2)`

(2) `if (C==0) PC = (PC & 0xfffffffc)+(sign_extend(disp24)<< 2)`

where :

```
#define sign_extend(x)  (((signed)((x)<< 8))>> 8)
```

Description

BNC branches to the specified label when the condition bit, C, is 0

There are two instruction formats; which allows software, such as an assembler, the decision on the better format.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0111	1101	disp8	BNC disp8

1111	1101	disp24	BNC disp24

BNE

Instruction

```
BNE Rsrc1,Rsrc2,Label
```

Function

```
if (Rsrc1!=Rsrc2) PC = (PC & 0xfffffffc)+(((signed short)disp16)<< 2)
```

Description

BNE branches to the specified label when Rsrc1 is not equal to Rsrc2

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1011	src1	0001	src2	disp16	BNE Rsrc1,Rsrc2,disp16

BNEZ

Instruction

```
BNEZ Rsrc,Label
```

Function

```
if (Rsrc!=0) PC = (PC & 0xfffffffc)+(((signed short)disp16)<< 2)
```

Description

BNEZ branches to the specified label when Rsrc is not equal to zero

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1011	0000	1001	src	disp16	BNEZ Rsrc,disp16

BRA

Instruction

```
BRA Label
(1) BRA disp8
(2) BRA disp24
```

Function

```
(1) PC = (PC & 0xfffffffc)+(((signed char)disp8)<< 2)
(2) PC = (PC & 0xfffffffc)+(sign_extend(disp24)<< 2)
```

where

```
#define sign_extend(x)  (((signed)((x)<< 8))>> 8)
```

Description

BRA branches unconditionally to the address specified by the label

There are two instruction formats; which allows software, such as an assembler, the decision on the better format.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0111	1111	disp8	BRA disp8

1111	1111	disp24	BRA disp24

CMP

Instruction

```
CMP Rsrc1,Rsrc2
```

Function

```
C = ((signed)Rsrc1 < (signed)Rsrc2) ? 1:0
```

Description

The condition bit, C, is set when Rsrc1 is less than Rsrc2. The operands are treated as signed 32-bit values.

Exceptions

None

Encoding

0000	src1	0100	src2	CMP Rsrc1,Rsrc2

CMPI

Instruction

```
CMPI Rsrc,#imm16
```

Function

```
C = ((signed)Rsrc < (signed short)imm16) ? 1:0
```

Description

The condition bit, C, is set when Rsrc is less than 16-bit immediate value. The operands are treated as signed 32-bit values. The immediate value is sign-extended to 32-bit before the operation.

Exceptions

None

Encoding

1000	0000	0100	src	imm16	CMPI Rsrc,#imm16

CMPU

Instruction

```
CMPU Rsrc1,Rsrc2
```

Function

```
C = ((unsigned)Rsrc1 < (unsigned)Rsrc2) ? 1:0
```

Description

The condition bit, C, is set when Rsrc1 is less than Rsrc2. The operands are treated as unsigned 32-bit values.

Exceptions

None

Encoding

0000	src1	0101	src2	CMPU Rsrc1,Rsrc2

CMPUI

Instruction

```
CMPUI Rsrc,#imm16
```

Function

```
C = ((unsigned)Rsrc < (unsigned)((signed short)imm16)) ? 1:0
```

Description

The condition bit, C, is set when Rsrc is less the than 16-bit immediate value. The operands are treated as unsigned 32-bit values. The immediate value is sign-extended the 32-bit before the operation.

Exceptions

None

Encoding

1000	0000	0101	src	imm16	CMPUI Rsrc,#imm16

DIV

Instruction

```
DIV Rdest,Rsrc
```

Function

```
Rdest = (signed)Rdest / (signed)Rsrc
```

Description

DIV divides Rdest by Rsrc and puts the quotient in Rdest

The operands are treated as signed 32-bit values and the result is rounded toward zero.

The condition bit, C, is unchanged.

When Rsrc is zero, Rdest is unchanged.

Exceptions

None

Encoding

1001	dest	0000	src	0	DIV Rdest,Rsrc

DIVU

Instruction

```
DIVU Rdest,Rsrc
```

Function

```
Rdest = (unsigned)Rdest / (unsigned)Rsrc
```

Description

DIVU divides Rdest by Rsrc and puts the quotient in Rdest

The operands are treated as unsigned 32-bit values and the result is rounded toward zero.

The condition bit, C, is unchanged.

When Rsrc is zero, Rdest is unchanged.

Exceptions

None

Encoding

1001	dest	0001	src	0	DIVU Rdest,Rsrc

JL

Instruction

```
JL Rsrc
```

Function

```
R14 = (PC & 0xfffffffc)+4
PC = Rsrc & 0xfffffffc
```

Description

JL unconditionally jumps to the location specified by Rsrc and puts the return address in R14

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0001	1110	1100	src	JL Rsrc

JMP

Instruction

`JMP Rsrc`

Function

`PC = Rsrc & 0xfffffffc`

Description

JMP unconditionally jumps to the location specified by Rsrc

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0001	1111	1100	src	JMP Rsrc

LD

Instruction

(1) `LD Rdest,@Rsrc`

(2) `LD Rdest,@Rsrc+`

(3) `LD Rdest,@(disp16,Rsrc)`

Function

(1) `Rdest = *(int *)Rsrc`

(2) `Rdest = *(int *)Rsrc, Rsrc += 4`

(3) `Rdest = *(int *)(Rsrc + (signed short)disp16)`

Description

(1) The contents of the word at the memory location specified by Rsrc are loaded into Rdest

(2) The contents of the word at the memory location specified by Rsrc are loaded into Rdest

Rsrc is post incremented by 4. When Rsrc and Rdest are same register, the result of the operation is undefined.

(3) The contents of the word at the memory location specified by Rsrc and the 16-bit displacement are loaded into Rdest

The displacement value is sign-extended before the address calculation.

The condition bit C is unchanged.

Exceptions

Address Exception

Encoding

0010	dest	1100	src	`LD Rdest,@Rsrc`

0010	dest	1110	src	`LD Rdest,@Rsrc+`

1010	dest	1100	src	disp16	`LD Rdest,@(disp16,Rsrc)`

LD24

Instruction

```
LD24 Rdest,#imm24
```

Function

```
Rdest = imm24 & 0x00ffffff
```

Description

LD24 loads the 24-bit immediate value into Rdest

The immediate value is zero-extended.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1110	dest	imm24	LD24 Rdest,#imm24

LDB

Instruction

(1) `LDB Rdest,@Rsrc`

(2) `LDB Rdest,@(disp16,Rsrc)`

Function

(1) `Rdest = *(signed char *)Rsrc`

(2) `Rdest = *(signed char *)(Rsrc + (signed short)disp16)`

Description

(1) LDB sign-extends the contents of the byte at the memory location specified by Rsrc and loads them into Rdest.

(2) LDB sign-extends the contents of the byte at the memory location specified by Rsrc and the 16-bit displacement and loads them into Rdest

The displacement value is sign-extended before the address calculation.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0010	dest	1000	src	LDB Rdest,@Rsrc

1010	dest	1000	src	disp16	LDB Rdest,@(disp16,Rsrc)

LDH

Instruction

(1) `LDH Rdest,@Rsrc`

(2) `LDH Rdest,@(disp16,Rsrc)`

Function

(1) `Rdest = *(signed short *)Rsrc`

(2) `Rdest = *(signed short *)(Rsrc + (signed short)disp16)`

Description

(1) LDH sign-extends the contents of the halfword at the memory location specified by Rsrc and loads them into Rdest.

(2) LDH sign-extends the contents of the halfword at the memory location specified by Rsrc with the 16-bit displacement and loads them into Rdest

The displacement value is sign-extended before the address calculation.

The condition bit, C, is unchanged.

Exceptions

Address Exception

Encoding

0010	dest	1010	src	LDH Rdest,@Rsrc

1010	dest	1010	src	disp16	LDH Rdest,@(disp16,Rsrc)

LDI

Instruction

(1) `LDI Rdest,#imm8`

(2) `LDI Rdest,#imm16`

Function

(1) `Rdest = (signed char)imm8`

(2) `Rdest = (signed short)imm16`

Description

(1) LDI loads the 8-bit immediate value into Rdest

The immediate value is sign-extended.

(2) LDI loads the 16-bit immediate value into Rdest

The immediate value is sign-extended.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0110	dest	imm8	LDI Rdest,#imm8

1001	dest	1111	0000	imm16	LDI Rdest,#imm16

LDUB

Instruction

(1) `LDUB Rdest,@Rsrc`

(2) `LDUB Rdest,@(disp16,Rsrc)`

Function

(1) `Rdest = *(unsigned char *)Rsrc`

(2) `Rdest = *(unsigned char *)(Rsrc + (signed short)disp16)`

Description

(1) LDUB zero-extends the contents of the byte at the memory location specified by Rsrc and loads them into Rdest

(2) LDUB zero-extends the contents of the byte at the memory location specified by Rsrc with the 16-bit displacement and loads them into Rdest

The displacement value is sign-extended before address calculation.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0010	dest	1001	src	LDUB Rdest,@Rsrc

1010	dest	1001	src	disp16	LDUB Rdest,@(disp16,Rsrc)

LDUH

Instruction

(1) `LDUH Rdest,@Rsrc`

(2) `LDUH Rdest,@(disp16,Rsrc)`

Function

(1) `Rdest = *(unsigned short *)Rsrc`

(2) `Rdest = *(unsigned short *)(Rsrc + (signed short)disp16)`

Description

(1) LDUH zero-extends the contents of the halfword at the memory location specified by Rsrc and loads them into Rdest

(2) LDUH zero-extends the contents of the halfword at the memory location specified by Rsrc with the 16-bit displacement and loads them into Rdest

The displacement value is sign-extended before the address calculation.

The condition bit, C, is unchanged.

Exceptions

Address Error

Encoding

0010	dest	1011	src		LDUH Rdest,@Rsrc
1010	dest	1011	src	disp16	LDUH Rdest,@(disp16,Rsrc)

LOCK

Instruction

```
LOCK Rdest,@Rsrc
```

Function

```
LOCK = 1, Rdest = *(int *)Rsrc
```

Description

The contents of the word at the memory location specified by Rsrc are loaded into Rdest.

This instruction, in addition to doing a simple load, has the effect of setting a LOCK bit.

When the LOCK bit is 1, external bus master access is not accepted.

The LOCK bit is cleared by executing UNLOCK instruction.

* The LOCK bit is CPU internal signal. The user cannot access this bit.

The condition bit, C, is unchanged.

Exceptions

Address Exception

Encoding

0010	dest	1101	src

`LOCK Rdest,@Rsrc`

MACHI

Instruction

```
MACHI Rsrc1,Rsrc2
```

Function

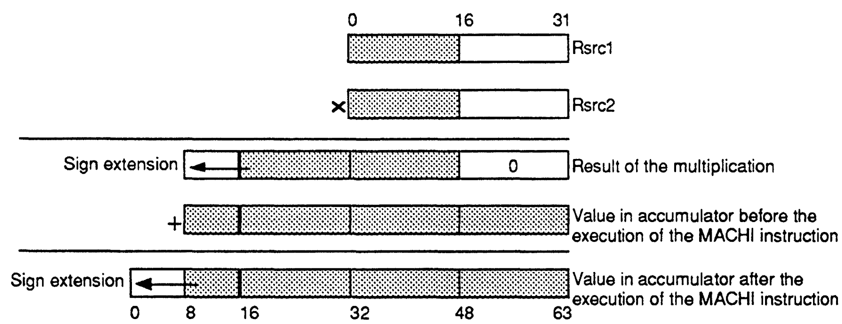
```
accumulator += ((signed)(Rsrc1 & 0xffff0000) * (signed short)(Rsrc2
>> 16))
```

Description

MACHI multiplies the high-order 16 bits of Rsrc1 and the high-order 16 bits of Rsrc2, then adds the result to the low-order 56 bits in the accumulator

However, the LSB of the result of the multiplication is aligned with bit 47 in the accumulator, and the portion corresponding to bits 8 through 15 of the accumulator is sign-extended before the addition. The result of the addition is stored in the accumulator. The high-order 16 bits of Rsrc1 and Rsrc2 are treated as signed values.

The condition bit, C, is unchanged.



Exceptions

None

Encoding

0011	src1	0100	src2	MACHI Rsrc1,Rsrc2

MACLO

Instruction

```
MACLO Rsrc1,Rsrc2
```

Function

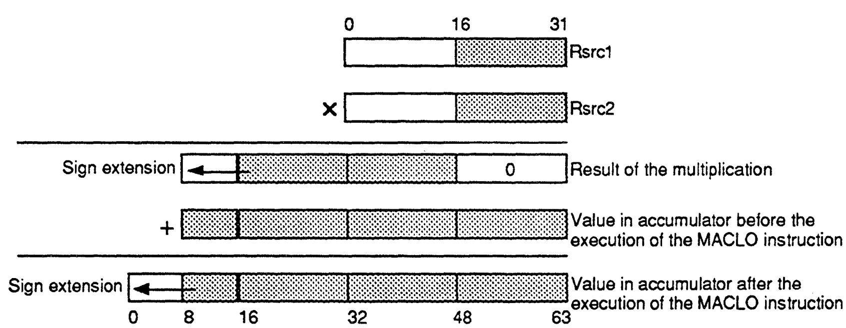
```
accumulator += ((signed)(Rsrc1 << 16) * (signed short)Rsrc2)
```

Description

MACLO multiplies the low-order 16 bits of Rsrc1 and the low-order 16 bits of Rsrc2, then adds the result to the low order 56 bits in the accumulator

However, the LSB of the result of the multiplication is aligned with bit 47 in the accumulator, and the portion corresponding to bits 8 through 15 of the accumulator is sign extended before the addition. The result of the addition is stored in the accumulator. The low-order 16 bits of Rsrc1 and Rsrc2 are treated as signed values.

The condition bit, C, is unchanged.



Exceptions

None

Encoding

0011	src1	0101	src2	MACLO Rsrc1,Rsrc2

MACWHI

Instruction

```
MACWHI Rsrc1,Rsrc2
```

Function

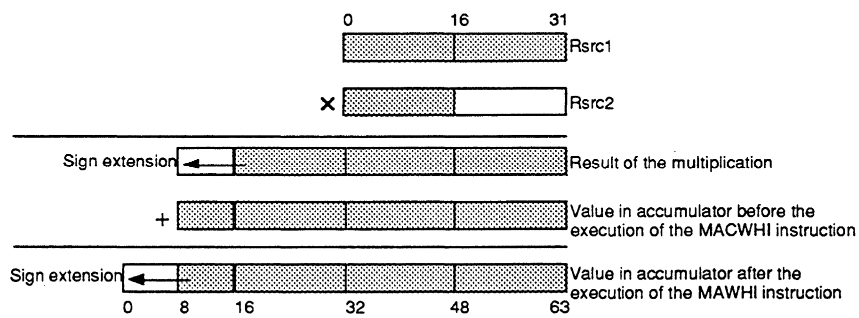
```
accumulator += ((signed)Rsrc1 * (signed short)(Rsrc2 >> 16))
```

Description

MACWHI multiplies the 32 bits of Rsrc1 and the high-order 16 bits of Rsrc2, then adds the result to the low-order 56 bits in the accumulator

However, the LSB of the result of the multiplication is aligned with the LSB of the accumulator, and the portion corresponding to bits 8 through 15 of the accumulator is sign extended before the addition. The result of the addition is stored in the accumulator. The 32 bits of Rsrc1 and Rsrc2 are treated as signed values.

The condition bit, C, is unchanged.



Exceptions

None

Encoding

0011	src1	0110	src2	MACWHI Rsrc1,Rsrc2

MACWLO

Instruction

```
MACWLO Rsrc1,Rsrc2
```

Function

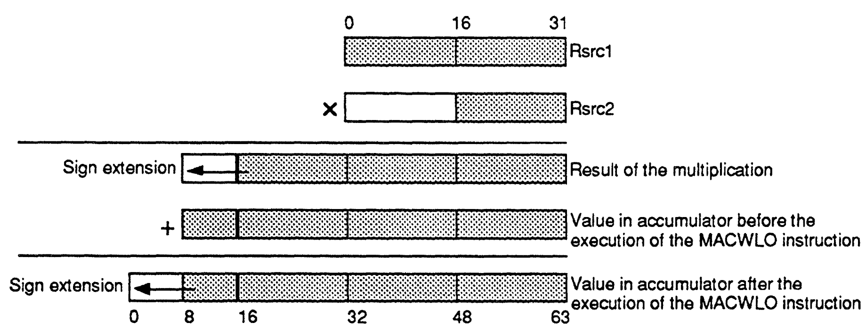
```
accumulator += ((signed)Rsrc1 * (signed short)Rsrc2)
```

Description

MACWLO multiplies the 32 bits of Rsrc1 and the low-order 16 bits of Rsrc2, then adds the result to the low-order 56 bits in the accumulator

However, the LSB of the result of the multiplication is aligned with the LSB of the accumulator, and the portion corresponding to bits 8 through 15 of the accumulator is sign-extended before the addition. The result of the addition is stored in the accumulator. The 32 bits Rsrc1 and the low-order 16 bits of Rsrc2 are treated as signed values.

The condition bit, C, is unchanged.



Exceptions

None

Encoding

0011	src1	0111	src2	MACWLO Rsrc1,Rsrc2

MUL

Instruction

```
MUL Rdest,Rsrc
```

Function

```
{ signed64bit tmp
tmp = (signed64bit)Rdest * (signed64bit)Rsrc
Rdest = (int)tmp
}
```

Description

MUL multiplies Rdest by Rsrc and puts the result in Rdest

The operands are treated as signed values.

The condition bit, C, is unchanged. The contents of the accumulator may be broken by this instruction.

Exceptions

None

Encoding

0001	dest	0110	src	MUL Rdest,Rsrc

MULHI

Instruction

```
MULHI Rsrc1,Rsrc2
```

Function

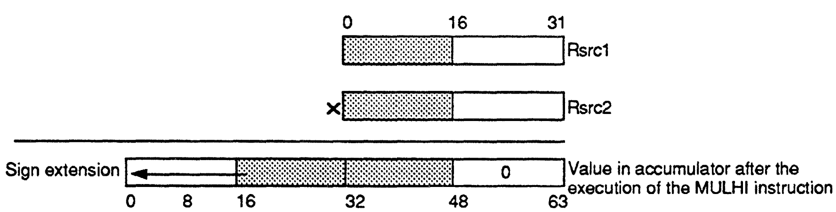
```
accumulator = ((signed)(Rsrc1 & 0xffff0000) * (signed short)(Rsrc2
>> 16))
```

Description

MULHI multiplies the high-order 16 bits of Rsrc1 and the high-order 16 bits of Rsrc2, and stores the result in the accumulator

However, the LSB of the result of the multiplication is aligned with bit 47 in the accumulator, and the portion corresponding to bits 0 through 15 of the accumulator is sign extended. Bits 48 through 63 of the accumulator are cleared (zero). The high-order 16 bits of Rsrc1 and Rsrc2 are treated as signed values.

The condition bit, C, is unchanged.



Exceptions

None

Encoding

0011	src1	0000	src2	MULHI Rsrc1,Rsrc2

MULLO

Instruction

```
MULLO Rsrc1,Rsrc2
```

Function

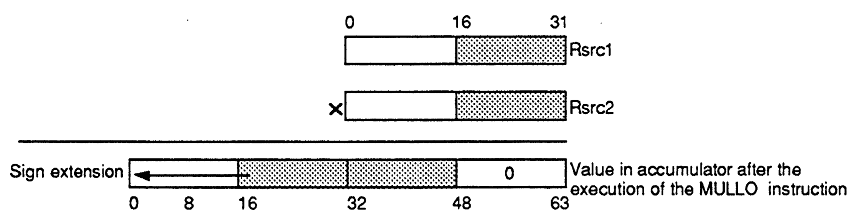
```
accumulator = ((signed)(Rsrc1 << 16) * (signed short)Rsrc2)
```

Description

MULLO multiplies the low-order 16 bits of Rsrc1 and the low-order 16 bits of Rsrc2, and stores the result in the accumulator

However, the LSB of the result of the multiplication is aligned with bit 47 in the accumulator, and the portion corresponding to bits 0 through 15 of the accumulator is sign extended. Bits 48 through 63 of the accumulator are cleared (zero). The low-order 16 bits of Rsrc1 and Rsrc2 are treated as signed values.

The condition bit, C, is unchanged.



Exceptions

None

Encoding

0011	src1	0001	src2	MULLO Rsrc1,Rsrc2

MULWHI

Instruction

```
MULWHI Rsrc1,Rsrc2
```

Function

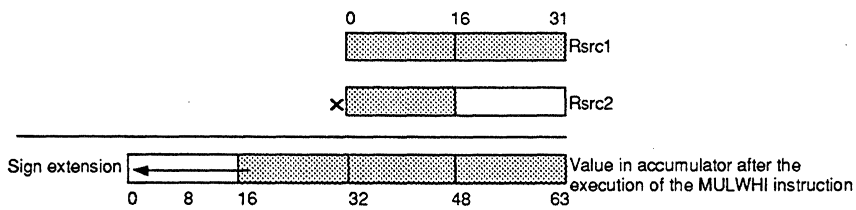
```
accumulator = ((signed)Rsrc1 * (signed short)(Rsrc2 >> 16))
```

Description

MULWHI multiplies the 32 bits of Rsrc1 and the high-order 16 bits of Rsrc2, and stores the result in the accumulator.

However, the LSB of the result of the multiplication is aligned with the LSB of the accumulator, and the portion corresponding to bits 0 through 15 of the accumulator is sign extended. The 32 bits of Rsrc1 and high-order 16 bits of Rsrc2 are treated as signed values.

The condition bit, C, is unchanged.



Exceptions

None

Encoding

0011	src1	0010	src2

MULWHI Rsrc1,Rsrc2

MULWLO

Instruction

```
MULWLO Rsrc1,Rsrc2
```

Function

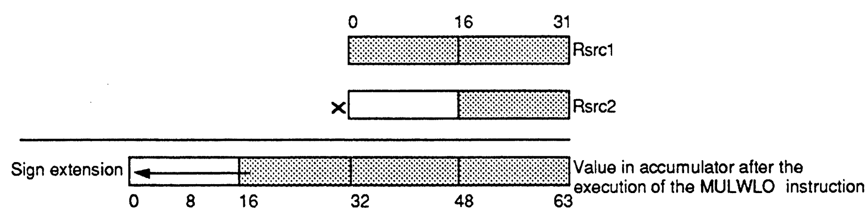
```
accumulator = ((signed)Rsrc1 * (signed short)Rsrc2)
```

Description

MULWLO multiplies the 32 bits of Rsrc1 and the low-order 16 bits of Rsrc2, and stores the result in the accumulator

However, the LSB of the result of the multiplication is aligned with the LSB of the accumulator, and the portion corresponding to bits 0 through 15 of the accumulator is sign extended. The 32 bits of Rsrc1 and low-order 16 bits of Rsrc2 are treated as signed values.

The condition bit, C, is unchanged.



Exceptions

None

Encoding

0011	src1	0011	src2	MULWLO Rsrc1,Rsrc2

MV

Instruction

`MV Rdest,Rsrc`

Function

`Rdest = Rsrc`

Description

MV moves the contents of Rsrc to the destination register Rdest.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0001	dest	1000	src	MV Rdest,Rsrc

MVFACHI

Instruction

```
MVFACHI Rdest
```

Function

```
Rdest = (int)(accumulator >> 32)
```

Description

MVFACHI moves the high-order 32-bit data in the accumulator to the destination register Rdest.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0101	dest	1111	0000	MVFACHI Rdest

MVFACLO

Instruction

```
MVFACLO Rdest
```

Function

```
Rdest = (int)accumulator
```

Description

MVFACLO moves the low-order 32-bit data in the accumulator to the destination register Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0101	dest	1111	0001	MVFACLO Rdest

MVFACMI

Instruction

```
MVFACMI Rdest
```

Function

```
Rdest = (int)(accumulator >> 16)
```

Description

MVFACMI moves the bit16-bit47 data in the accumulator to the destination register Rdest.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0101	dest	1111	0010	MVFACMI Rdest

MVFC

Instruction

```
MVFC Rdest,CRsrc
```

Function

```
Rdest = CRsrc
```

Description

MVFC moves the contents of the control register CRsrc to the destination register Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0001	dest	1001	src

MVFC Rdest,CRsrc

MVTACHI

Instruction

```
MVTACHI Rsrc
```

Function

```
accumulator[0:31] = Rsrc
```

Description

MVTACHI moves the contents of Rsrc to high-order 32-bit in the accumulator [0:31]

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0101	src	0111	0000	MVTACHI Rsrc

MVTACLO

Instruction

```
MVTACLO Rsrc
```

Function

```
accumulator[32:63] = Rsrc
```

Description

MVTACLO moves the contents of Rsrc to low-order 32-bit in the accumulator.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0101	src	0111	0001	MVTACLO Rsrc

MVTC

Instruction

```
MVTC Rsrc,CRdest
```

Function

```
CRdest = Rsrc
```

Description

MVTC moves the contents of Rsrc to the destination control register CRdest

If CRdest specifies the PSW(CR0), the condition bit, C, is changed; otherwise it is unchanged.

Exceptions

None

Encoding

0001	dest	1010	src	MVTC Rsrc,CRdest

NEG

Instruction

```
NEG Rdest,Rsrc
```

Function

```
Rdest = 0 - Rsrc
```

Description

NEG negates (changes the sign of) the contents of Rsrc, treated as a signed 32-bit value, and puts the result in Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0000	dest	0011	src	NEG Rdest,Rsrc

NOP

Instruction

```
NOP
```

Function

```
/* */
```

Description

This instruction does not change the state of the machine and takes one cycle to execute.

Exceptions

None

Encoding

0111	0000	0000	0000	NOP

NOT

Instruction

```
NOT Rdest,Rsrc
```

Function

```
Rdest = ~Rsrc
```

Description

NOT inverts each of the bits of the contents of Rsrc and puts the result in Rdest.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0000	dest	1011	src	NOT Rdest,Rsrc

OR

Instruction

```
OR Rdest,Rsrc
```

Function

```
Rdest = Rdest | Rsrc
```

Description

OR computes the logical OR of the corresponding bits of Rdest and Rsrc and puts the result in Rdest.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0000	dest	1110	src	OR Rdest,Rsrc

OR3

Instruction

`OR3 Rdest,Rsrc,#imm16`

Function

`Rdest = Rsrc | (unsigned short)imm16`

Description

OR3 computes the logical OR of the corresponding bits of Rsrc and the 16-bit immediate value, which is zero-extended from 16-bits to 32-bits, and puts the result in Rdest.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1000	dest	1110	src	imm16	
					OR3 Rdest,Rsrc,#imm16

RAC

Instruction

```
RAC
```

Function

```
{ signed64bit tmp
if( 0x0000 3fff ffff 8000 =< accumulator )
  tmp = 0x0000 3fff ffff 8000
else if( accumulator =< 0xffff c000 0000 0000 )
  tmp = 0xffff c000 0000 0000
else {
  tmp = accumulator + 0x0000 0000 0000 4000
  tmp = tmp & 0xffff ffff ffff 8000
}
accumulator = tmp << 1
}
```

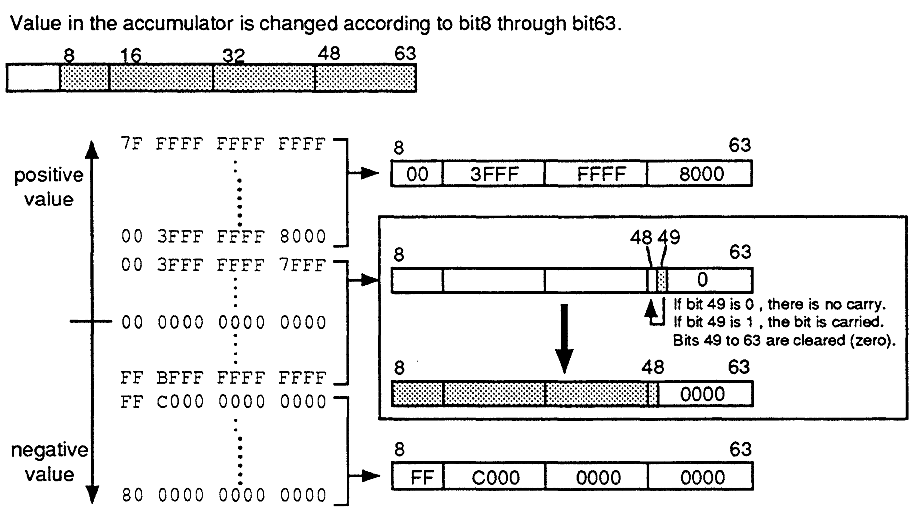
Description

RAC rounds the value in the accumulator to one word size and stores the result in the accumulator

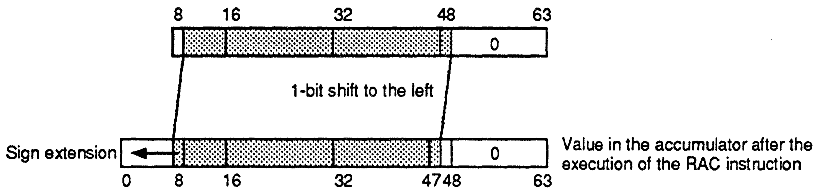
This instruction is executed in two steps, as shown following.

The condition bit, C, is unchanged.

Step1 :



Step2 :



Exceptions

None

Encoding

0101	0000	1001	0000	RAC

RACH

Instruction

```
RACH
```

Function

```
{ signed64bit tmp
if( 0x0000 3fff 8000 0000 =< accumulator )
  tmp = 0x0000 3fff 8000 0000
else if( accumulator =< 0xffff c000 0000 0000 )
  tmp = 0xffff c000 0000 0000
else {
  tmp = accumulator + 0x0000 0000 4000 0000
  tmp = tmp & 0xffff ffff 8000 0000
}
accumulator = tmp << 1
}
```

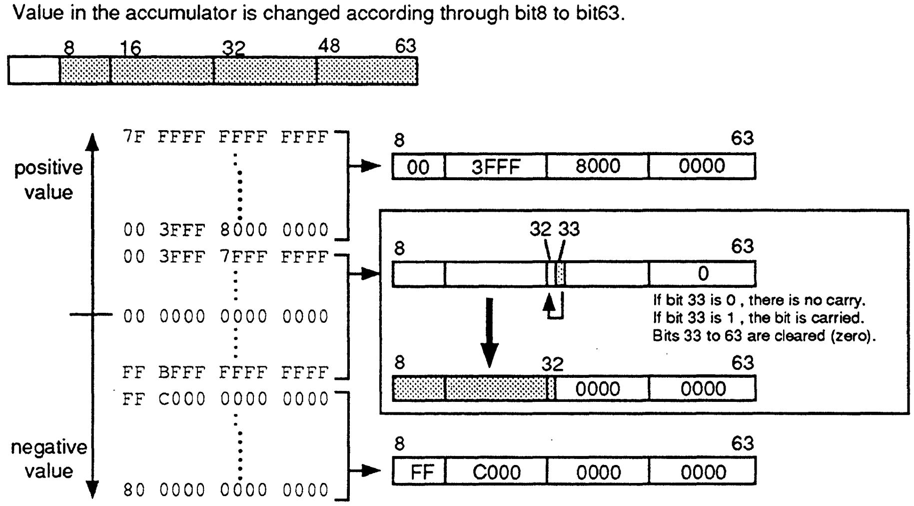
Description

RACH rounds the value in the accumulator to a halfword size and stores the result in the accumulator. This instruction is executed in two steps, as shown below.

The condition bit C is unchanged.

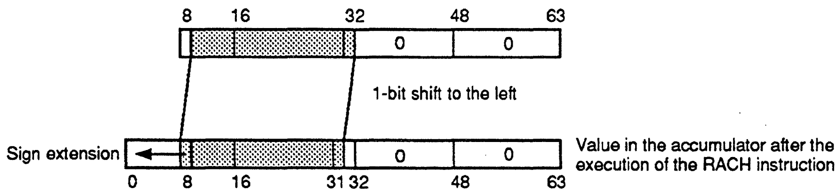
Step1:

Value in the accumulator is changed according through bit8 to bit63.



RACH

Step2:



Exceptions

None

Encoding

0101	0000	1000	0000

RACH

REM

Instruction

```
REM Rdest,Rsrc
```

Function

```
Rdest = (signed)Rdest % (signed)Rsrc
```

Description

REM divides Rdest by Rsrc and puts the quotient in Rdest. The operands are treated as signed 32-bit values

The quotient is rounded toward zero and the quotient takes the same sign as the dividend.

The condition bit, C, is unchanged.

When Rsrc is zero, Rdest is unchanged.

Exceptions

None

Encoding

1001	dest	0010	src	0	REM Rdest,Rsrc

REMU

Instruction

```
REMU Rdest,Rsrc
```

Function

```
Rdest = (unsigned)Rdest % (unsigned)Rsrc
```

Description

REMU divides Rdest by Rsrc and puts the quotient in Rdest

The operands are treated as unsigned 32-bit values.

The condition bit C is unchanged.

When Rsrc is zero, Rdest is unchanged.

Exceptions

None

Encoding

1001	dest	0011	src	0	REMU Rdest,Rsrc

RTE

Instruction

```
RTE
```

Function

```
SM = BSM
IE = BIE
C = BC
PC = BPC & 0xfffffffc
```

Description

RTE restores the BSM, BIE and BC bits of the PSW into the SM, IE and C bits, and jumps to the location specified by BPC

Exceptions

None

Encoding

0001	0000	1101	0110	RTE

SETH

Instruction

```
SETH Rdest,#imm16
```

Function

```
Rdest = (short)imm16 << 16
```

Description

SETH loads the immediate value into the sixteen most significant bits of Rdest

The sixteen least significant bits are set to zero.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1101	dest	1100	0000	imm16	SETH Rdest,#imm16

SLL

Instruction

```
SLL Rdest,Rsrc
```

Function

```
Rdest = Rdest << (Rsrc & 31)
```

Description

SLL left-shifts Rdest by the number of bits specified by Rsrc, shifts zeroes into the least significant bit and puts the result in Rdest

Only the five least significant bits of Rsrc are used as the shift amount.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0001	dest	0100	src	SLL Rdest,Rsrc

SLL3

Instruction

```
SLL3 Rdest,Rsrc,#imm16
```

Function

```
Rdest = Rsrc << (imm16 & 31)
```

Description

SLL3 left-shifts Rsrc by the amount specified by the 16-bit immediate value, shifts zeroes into the least significant bits and puts the result in Rdest

Only the five least significant bits are used as the shift amount.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1001	dest	1100	src	imm16	
					SLL3 Rdest,Rsrc,#imm16

SLLI

Instruction

```
SLLI Rdest,#imm5
```

Function

```
Rdest = Rdest << (imm5 & 31)
```

Description

SLLI left-shifts Rdest by the amount specified by the 5-bit immediate value, shifts zeroes into the least significant bits and puts the result in Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0101	dest	010	imm5	SLLI Rdest,#imm5

SRA

Instruction

```
SRA Rdest,Rsrc
```

Function

```
Rdest = (signed)Rdest >> (Rsrc & 31)
```

Description

SRA right-shifts Rdest by the amount specified by Rsrc, replicates the sign bit of Rdest in the most significant position and puts the result in Rdest

Only the five least significant bits are used as the shift amount.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0001	dest	0010	src	SRA Rdest,Rsrc

SRA3

Instruction

```
SRA3 Rdest,Rsrc,#imm16
```

Function

```
Rdest = (signed)Rsrc >> (imm16 & 31)
```

Description

SRA3 right-shifts Rsrc by the amount specified by the 16-bit immediate value, replicates the sign bit of Rsrc in the most significant bit and puts the result in Rdest

Only the five least significant bits are used as the shift amount.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1001	dest	1010	src	imm16	SRA3 Rdest,Rsrc,#imm16

SRAI

Instruction

```
SRAI Rdest,#imm5
```

Function

```
Rdest = (signed)Rdest >> (imm5 & 31)
```

Description

SRAI right-shifts Rdest by the amount specified by the 5-bit immediate value, replicates the sign bit of Rdest in the most significant position and puts the result in Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0101	dest	001	imm5	SRAI Rdest,#imm5

SRL

Instruction

```
SRL Rdest,Rsrc
```

Function

```
Rdest = (unsigned)Rdest >> (Rsrc & 31)
```

Description

SRL right-shifts Rdest by the amount specified by Rsrc, shifts zeroes into the most significant bits and puts the result in Rdest

Only the five least significant bits of Rsrc are used as the shift amount.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0001	dest	0000	src	SRL Rdest,Rsrc

SRL3

Instruction

```
SRL3 Rdest,Rsrc,#imm16
```

Function

```
Rdest = (unsigned)Rsrc >> (imm16 & 31)
```

Description

SRL3 right-shifts Rsrc by the amount specified by the 16-bit immediate value, shifts zeroes into the most significant bits and puts the result in Rdest. Only the five least significant bits of the immediate value are used as the shift amount.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1001	dest	1000	src	imm16	SRL3 Rdest,Rsrc,#imm16

SRLI

Instruction

`SRLI Rdest,#imm5`

Function

`Rdest = (unsigned)Rdest >> (imm5 & 31)`

Description

SRLI right-shifts Rdest by the amount specified by the 5-bit immediate value, shifts zeroes into the most significant bits and puts the result in Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0101	dest	000	imm5	SRLI Rdest,#imm5

ST

Instruction

(1) `ST Rsrc1,@Rsrc2`

(2) `ST Rsrc1,@+Rsrc2`

(3) `ST Rsrc1,@-Rsrc2`

(4) `ST Rsrc1,@(disp16,Rsrc2)`

Function

(1) `*(int *)Rsrc2 = Rsrc1`

(2) `Rsrc2 += 4, *(int *)Rsrc2 = Rsrc1`

(3) `Rsrc2 -= 4, *(int *)Rsrc2 = Rsrc1`

(4) `*(int *)(Rsrc2 + (signed short)disp16) = Rsrc1`

Description

(1) ST stores the contents of Rsrc1 in the memory location specified by Rsrc2

(2) ST pre-increments Rsrc2 by 4 and stores the contents of Rsrc1 in the memory location specified by the resultant Rsrc2

(3) ST pre-decrements Rsrc2 by 4 and stores the contents of Rsrc1 in the memory location specified by the resultant Rsrc2

(4) ST stores the contents of Rsrc1 in the memory location specified by Rsrc and the 16-bit displacement

The displacement value is sign-extended before the address calculation.

The condition bit, C, is unchanged.

Exceptions

Address Exception

Encoding

0010	src1	0100	src2	ST Rsrc1,@Rsrc2

0010	src1	0110	src2	ST Rsrc1,@+Rsrc2

0010	src1	0111	src2	ST Rsrc1,@-Rsrc2

1010	src1	0100	src2	disp16	ST Rsrc1,@(disp16,Rsrc2)

STB

Instruction

(1) `STB Rsrc1,@Rsrc2`

(2) `STB Rsrc1,@(disp16,Rsrc2)`

Function

(1) `*(char *)Rsrc2 = Rsrc1`

(2) `*(char *)(Rsrc2 + (signed short)disp16) = Rsrc1`

Description

(1) STB stores the contents of the least significant byte of Rsrc1 in the memory location specified by Rsrc2

(2) STB stores the contents of the least significant byte of Rsrc1 in the memory location specified by Rsrc with the 16-bit displacement

The displacement value is sign-extended before the address calculation.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0010	src1	0000	src2	`STB Rsrc1,@Rsrc2`

1010	src1	0000	src2	disp16	`STB Rsrc1,@(disp16,Rsrc2)`

STH

Instruction

(1) `STH Rsrc1,@Rsrc2`

(2) `STH Rsrc1,@(disp16,Rsrc2)`

Function

(1) `*(short *)Rsrc2 = Rsrc1`

(2) `*(short *)(Rsrc2 + (signed short)disp16) = Rsrc1`

Description

(1) STH stores the contents of the least significant halfword of Rsrc1 in the memory location specified by Rsrc2.

(2) STH stores the contents of the least significant halfword of Rsrc1 in the memory location specified by Rsrc with the 16-bit displacement

The displacement value is sign-extended before the address calculation.

The condition bit, C, is unchanged.

Exceptions

Address Exception

Encoding

0010	src1	0010	src2	STH Rsrc1,@Rsrc2

1010	src1	0010	src2	disp16	STH Rsrc1,@(disp16,Rsrc2)

SUB

Instruction

```
SUB Rdest,Rsrc
```

Function

```
Rdest = Rdest - Rsrc
```

Description

SUB subtracts Rsrc from Rdest and puts the result in Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0000	dest	0010	src	SUB Rdest,Rsrc

SUBV

Instruction

```
SUBV Rdest,Rsrc
```

Function

```
Rdest = (signed)Rdest - (signed)Rsrc
C = overflow ? 1:0
```

Description

SUBV subtracts Rsrc from Rdest and puts the result in Rdest

The condition bit, C, is set when the subtraction results in a two's-complement overflow; otherwise, C is cleared.

Exceptions

None

Encoding

0000	dest	0000	src	SUBV Rdest,Rsrc

SUBX

Instruction

```
SUBX Rdest,Rsrc
```

Function

```
Rdest = (unsigned)Rdest - (unsigned)Rsrc - C
C = borrow ? 1:0
```

Description

SUBX subtracts (Rsrc)+(the condition bit C) from Rdest and puts the result in Rdest

The condition bit, C, is set when the subtraction result can't be represented by a 32-bit unsigned integer; otherwise it is cleared.

Exceptions

None

Encoding

0000	dest	0001	src	SUBX Rdest,Rsrc

TRAP

Instruction

```
TRAP #imm4
```

Function

```
call_trap_handler(imm4)
```

Description

TRAP generates a trap

Exceptions

TRAP

Encoding

0001	0000	1111	imm4	TRAP #imm4

UNLOCK

Instruction

```
UNLOCK Rsrc1,@Rsrc2
```

Function

```
if ( LOCK == 1 ) { *(int *)Rsrc2 = Rsrc1}
LOCK = 0
```

Description

UNLOCK stores the contents of Rsrc1 in the memory location specified by Rsrc2 when the LOCK bit is 1

If the LOCK bit is 0, the store operation is not executed.

This instruction clears the LOCK bit.

* The LOCK bit is CPU internal signal. The user cannot access this bit.

The condition bit, C, is unchanged.

Exceptions

Address Exception

Encoding

0010	src1	0101	src2	UNLOCK Rsrc1,@Rsrc2

XOR

Instruction

```
XOR Rdest,Rsrc
```

Function

```
Rdest = Rdest ^ Rsrc
```

Description

XOR computes the logical XOR of the corresponding bits of Rdest and Rsrc and puts the result in Rdest

The condition bit, C, is unchanged.

Exceptions

None

Encoding

0000	dest	1101	src	XOR Rdest,Rsrc

XOR3

Instruction

```
XOR3 Rdest,Rsrc,#imm16
```

Function

```
Rdest = Rsrc ^ (unsigned short)imm16
```

Description

XOR3 computes the logical XOR of the corresponding bits of Rsrc and the 16-bit immediate value, with zero-extended from 16-bits through 32-bits and puts the result in Rdest.

The condition bit, C, is unchanged.

Exceptions

None

Encoding

1000	dest	1101	src	imm16	XOR3 Rdest,Rsrc,#imm16

Appendix B: I/O Registers

B.1 Registers for Internal Peripheral I/O

Fissures B.1 and B.2 are memory maps of the M32R internal I/O registers :

Logical Address	+0 address (b0 – b7)	+1 address (b8 – b15)	+2 address (b16 – b23)	+3 address (b24 – b31)
h'FFFFFFF4				MSPR
h'FFFFFFF8				MDRR
h'FFFFFFFC				MCCR

Figure B.1 Memory Map of the Internal Memory System Control Registers

Logical Address	+0 address (b0 – b7)	+1 address (b8 – b15)	+2 address (b16 – b23)	+3 address (b24 – b31)
h'FFFFFFE0				PPCR0
h'FFFFFFE4				PPCR1
h'FFFFFFE8				PPDR0
h'FFFFFFEC				PPDR1

Figure B.2 Memory Map of the Programmable Port Control Registers

B.2 Configuration of Internal I/O Registers

This section shows the bit configuration of each internal I/O register and uses the following conventions.

Term	Contents
b	Indicates the bit position (b0-b7)
Function	Shows the function that corresponds to each bit pattern × : indicates any value (0 or 1).
Ini	Shows bit value immediately after a reset has been cancelled. 0 : 0 (fixed) 1 : 1 (fixed) ? : unfixed
R	Shows read attribute o : read enabled 0 : fixed to 0 when reading × : read disabled 1 : fixed to 1 when reading
W	Shows write attribute o : write enabled × : write disabled (write requests ignored)

B.2.1 Cache Purge Control Register (MSPR)

b	Name	Function	Ini	R	W
0 | 6	(No mapping)		0	0	×
7	Purge specification bit	0 : No purge. 1 : Purge.	0	0	o

- Purge Specification Bit (b7)

  0 : The cache is not purged.

  1 : The cache is purged after writing the data in the cache to the DRAM.

B.2.2 DRAM Refresh Control Register (MDRR)

b	Name	Function	Ini	R	W
0–6	(No mapping)		0	0	×
7	Refresh mode bit	0 : Auto-refresh mode 1 : Self-refresh mode	0	o	o

- Refresh Mode Bit (b7)
  Sets refresh mode of the on-chip DRAM.

  0 : **auto-refresh mode**
  The memory controller performs a refresh at regular intervals.

  1 : **self-refresh mode**
  Converts the self-refresh mode

B.2.3 Memory Controller Configuration Register (MCCR)

b	Name	Function	Ini	R	W
0 | 4	(No mapping)		0	0	×
5	timing parameter bit	0 : Fast mode 1 : Slow mode	0	o	o
6 7	cache mode bits	b6 b7 0 X : Cache-off mode 1 0 : On-chip DRAM space shared cache mode 1 1 : External user space instruction cache mode	0	o	o

- Timing Parameter Bit (b5)
  The memory controller optimizes the timing of internal DRAM access according to the internal operating frequency.

  0 : Set to "0" when CLKIN is more than 8.3MHz.
  1 : Set to "1" when CLKIN is less than 8.3MHz.

||||| NOTE |||||

Note that operation cannot be guaranteed if b5 is "1" when CLKIN is more than 8.3MHz.

- Cache Mode Bits (b6,b7)
  Sets the on-chip SRAM cache mode

  0X : **cache-off mode**
  Stops cache operation

  10 : **on-chip DRAM space shared cache mode**
  Both instructions and data are cached when the internal DRAM space is accessed.

  11 : **external user space instruction cache mode**
  Instruction fetches are cached when the external user space is accessed.

B.2.4 Programmable Port Direction Control Registers (PPCR0, PPCR1)

b	Name	Function	Ini	R	W
0 | 6	(No mapping)		0	0	×
7	Port I/O direction specification bit	0 : Input mode 1 : Output mode	0	o	o

- Port I/O Direction Specification Bit (b7)

  0 : Input mode
  1 : Output mode

B.2.5 Programmable Port Data Registers (PPDR0, PPDR1)

b	Name	Function	Ini	R	W
0 | 6	(No mapping)		0	o	×
7	Port data bit	0 : Data = "0" 1 : Data = "1"	?	o	o

- Port Data Bit (b7)

  0 : port data = "0"
  1 : port data = "1"

Appendix C:

Pipeline Processing

C.1 Pipeline Stages

The M32R CPU consists of five pipeline stages.

- **IF stage (instruction fetch stage)**
  The instruction is fetched during the IF stage. There is an instruction queue and instructions are fetched until the queue is full regardless of the completion of decoding in the D stage.

- **D stage (decode stage)**
  Instruction decoding is performed during the first half of the D stage (DEC1). The instruction decoding (DEC2) is performed a register fetch (RF) in the second half of the time of the stage.

- **E stage (execution stage)**
  Operations and calculation addresses (OP) are performed during the E stage.

- **MEM stage (memory access stage)**
  Operand accesses (OA) are performed during the MEM stage. This stage is used only when executing load or store instructions.

- **WB stage (write back stage)**
  The results of operations and fetched data are written to the registers during the WB stage.

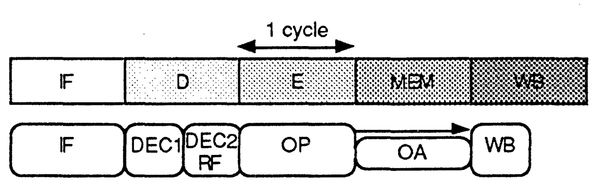


Figure C.1 Pipeline Stages

C.2 Actual Pipeline Processing

Because the MEM stage is used for Load/Store instructions, 5-stage pipeline processing is performed. However, the MEM stage is not used for other instructions, and only a 4-stage pipeline processing is performed.

- Load/Store instruction

IF	D	E	MEM	WB

If the cache is hit, the MEM stage is executed in one cycle. If missed, however, the MEM stage is executed for several cycles.

IF	D	E	MEM	···	MEM	WB

- Other instructions

IF	D	E	WB

The E stage is executed for several cycles in the case of multi-cycle instructions such as multiplication and division.

IF	D	E	E	···	E	WB

- The Stall of pipeline processing

In perfect pipeline processing, each stage is executed in one cycle. However, changes in the number of execution cycles in each stage and the execution of branch instructions mean that the pipeline processing may be stalled.

Case 1 : Execution of an instruction requiring multiple cycles in stage E.

DIV	r1,r2	IF	D	E	E	···	E	WB			
ADD	r3,r4		IF	D	stall	···	stall	E	WB		
ADD	r5,r6			IF	stall	···	stall	D	E	WB	
ADD	r7,r8				stall	···	stall	IF	D	E	WB

Case 2 : Operand access not completed in one cycle.

LD r1,@r2	IF	D	E	MEM	MEM	WB		
LD r3,@r4		IF	D	E	stall	MEM	WB	
ADD r5,r6			IF	D	stall	E	WB	
ADD r7,r8				IF	stall	D	E	WB

Case 3 : Branching occurs (Pipeline processing is not stalled if no branch occurs at a conditional branch instruction.)

Branch instruction	IF	D	E	WB					
		IF	D	IF	D	E	WB		
			IF	stall	IF	D	E	WB	
				stall	stall	IF	D	E	WB

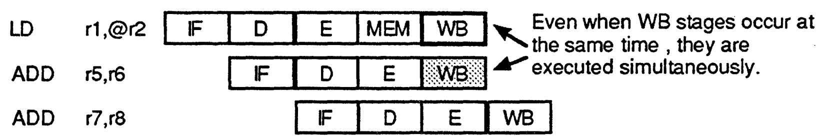
Case 4 : When a value read from memory is used by a subsequent instruction, stage E of the subsequent instruction is stalled until the load instruction completes the WB stage.

LD *r1*,@r2	IF	D	E	MEM	WB		
LD r3,*r1*		IF	D	stall	stall	E	WB

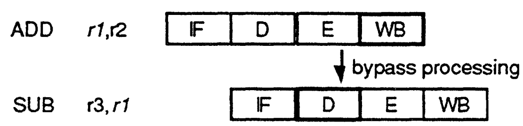
Case 5 : When reading R15 after writing the PSW.SM bit, the subsequent instruction is waiting for the E stage until the MVTC instruction complete the WB stage.

MVTC r1,*PSW*	IF	D	E	WB		
SUB r3,*r15*		IF	D	stall	E	WB

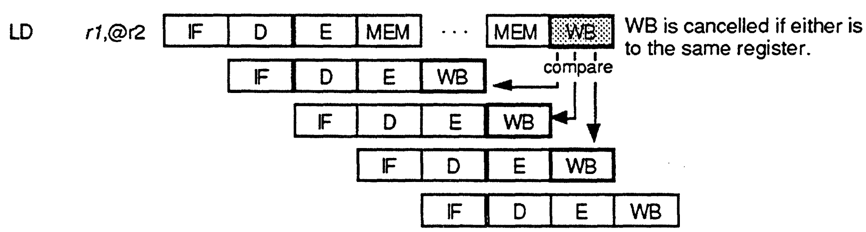
- Even when the WB stages of load and other instructions occur simultaneously, the values are written simultaneously to the registers and pipeline processing is not stalled.



- The bypass function prevents pipeline processing being stalled in the case of instructions for operations on the values of registers.



- Execution of the WB stage of the load instruction is cancelled if a subsequent instruction writes to the same register before execution of the load instruction has finished.



Appendix D: Instruction Execution Cycles

D.1 Number of Instruction Execution Cycles

The following shows the number of instruction execution cycles for each pipeline stage of each instruction type. Normally, the E stage is monitored as the instruction processing time, but, because of pipeline processing (See Appendix C) the processing times for other stages may effect the total instruction execution time. In particular, the IF, D, and E stages of a subsequent instruction are monitored after a branch has occurred.

- Load instruction (LDB, LDUB, LDH, LDUH, LD, LOCK)

IF	D	E	MEM	WB
R	1	1	R	1

- Store instruction (STB ,STH ,ST ,UNLOCK)

IF	D	E	MEM
R	1	1	W

- Multiply instruction (MUL)

IF	D	E	WB
R	1	3	1

- Divide/Remainder instruction (DIV, DIVU, REM, REMU)

IF	D	E	WB
R	1	37	1

- Other instructions

IF	D	E	WB
R	1	1	1

D.2 Number of Memory Access Cycles

This section shows the number of memory access cycles in the IF and MEM stages.

The values shown are the minimum number of cycles used by the CPU for memory access. They may not be equal to the numbers of cycles the memory and bus are actually used. In write access, for example, the CPU completes the MEM stage as soon as a value is written to the write buffer, but the value is actually written to memory afterwards. These values may increase depending on the state of memory and the bus when the CPU outputs a memory access request.

- R (read cycle)

When	Number of Cycles
In instruction queue	1
On-chip cache hit	1
On-chip cache miss and value read from on-chip DRAM	7 (Note1)
Cache off and value read from on-chip DRAM	6 (Note2)
Value read from external memory (byte-sized, halfword-sized)	10 (Note3)
Value read from external memory (word-sized)	11 (Note3)

(Note1) 4 cycles when the timing parameter bits of the memory controller configuration register (MCCR) are 1.

(Note2) 3 cycles when the timing parameter bits of MCCR is 1.

(Note3) When the external memory is accessed with no wait, because the internal and external clocks have different frequencies, the timing of access requests from the CPU may cause the values to vary.

- W (write cycle)

When	Number of Cycles
On-chip cache hit	1
On-chip cache miss and value write to on-chip DRAM	2
Cache off and value write to on-chip DRAM	2
Value write to external memory	2

Index

SYMBOLS

A

B

F

G

H

I

J

L

M

N

O

P

U

V

W

X

TABLES

FIGURES

MITSUBISHI SEMICONDUCTORS
M32R Family User's Manual (for 2MB On-chip DRAM Type)

First Edition January 12, 1996

Published by
Mitsubishi Electric Corporation, System LSI Laboratory