Section 2
Data Types

This section describes the hardware supported data types of the µPD70616 microprocessor. A total of eight separate fixed and variable length data types are supported by the µPD70616 microprocessor. Fixed length and variable length data types are distinguished by two major characteristics. Fixed length data types have a size characteristic that is constant and determined by the instruction opcodes. Variable length data types are dynamic data structures whose length can vary during program execution.

Another distinguishing characteristic is that the fixed length data types may reside in either the general purpose register set or in memory. Variable length data types can be represented in their entirety only as memory resident data types.

Fixed Length Data Types

- Integer
- Unsigned Integer
- Bit
- Floating Point
- Decimal

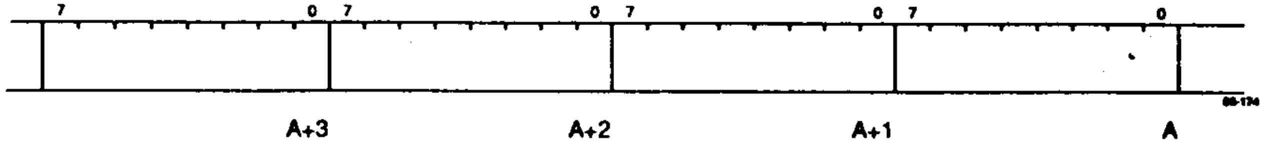
Variable Length Data Types

- Character String
- Bit String
- Bit Field

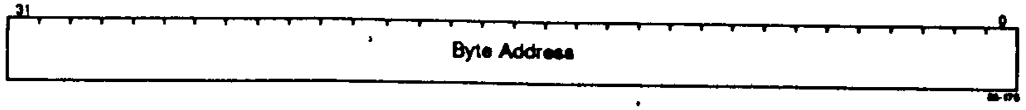
Also important is the organization of the data types. Data type organization describes the layout of a data type in a register or in memory. The µPD70616 architecture is flexible in that both byte and bit addressing modes are available. Byte addressing uses the byte (8-bits) as the basic addressing unit and supports the byte aligned data types. Bit addressing is a special addressing mode using the bit as the basic addressing unit and supports the bit aligned data types (bit fields and bit strings).

Byte Addressing

The addressing model for the µPD70616 virtual address space is based on the unit of a byte. The µPD70616 address space is viewed as a sequence of bytes starting from location 0 and continuing linearly to the location $2^{32} - 1$. All memory management, instruction fetches, stack operations and software debug operations use byte addresses.



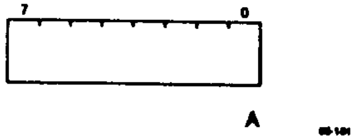
e addressing is also used for fixed length data types and for the variable length character string data type. These ta types share the characteristic that they are always aligned on a byte boundary, irregardless of the size of the data e. Since byte addresses are 32-bits wide, a byte address corresponds to the location of a byte anywhere within a ur gigabyte virtual address space.



e aligned memory operands are physically accessed using one of four access types, each of which can be located ywhere within the virtual address space without restriction. However, instruction throughput is optimized when an cess type is aligned on a boundary that is a multiple of its size in bytes.

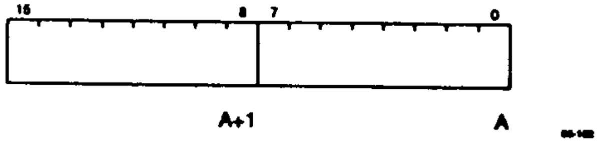
yte

byte consists of 8 contiguous bits starting on any byte boundary. The individual bits within a byte are labeled 0 to 7 h bit 0 designated as the LSB (least significant bit) and bit 7 as the MSB (most significant bit). A byte is completely ntified by its address.



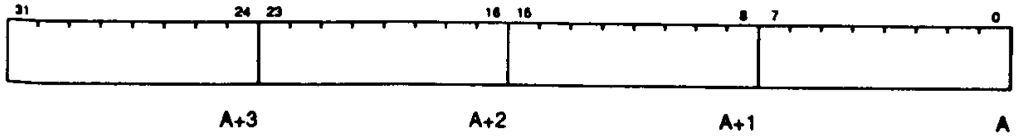
alfword

halfword consists of 16 contiguous bits starting on any byte boundary. The individual bits within a byte are labeled 0 15 with bit 0 designated as the LSB (least significant bit) and bit 15 as the MSB (most significant bit). A halfword cupies two contiguous bytes and is identified by the address of the low order byte.



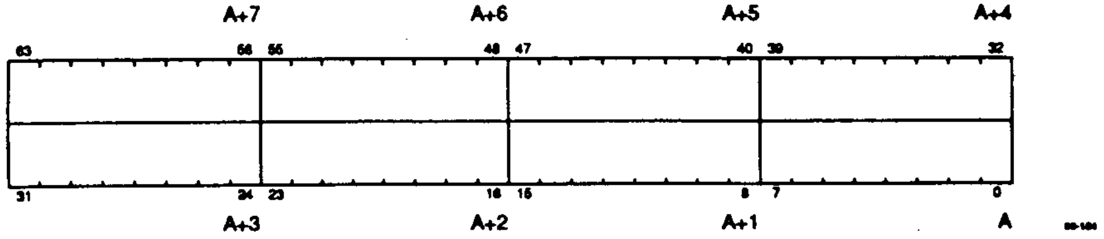
ord

word consists of 32 contiguous bits starting on any byte boundary. The individual bits within a byte are labeled 0 to with bit 0 designated as the LSB (least significant bit) and bit 31 as the MSB (most significant bit). A word occupies ur contiguous bytes and is identified by the address of the low order byte.

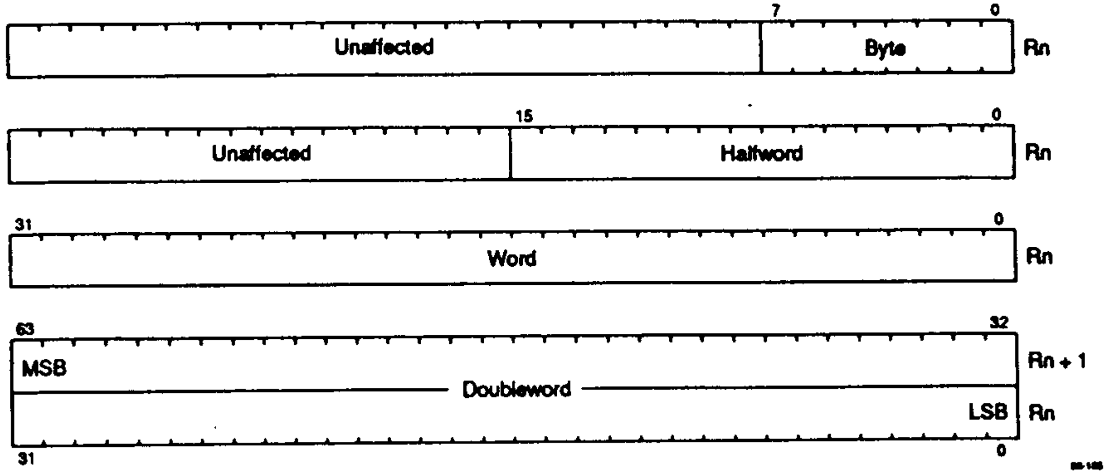


Doubleword

A doubleword consists of 64 contiguous bits starting on any byte boundary. The individual bits within a byte are labeled 0 to 63 with bit 0 designated as the LSB (least significant bit) and bit 63 as the MSB (most significant bit). A doubleword occupies eight contiguous bytes and is identified by the address of the low order byte.



Fixed length data types can also reside in one the µPD70616 general purpose registers. In place of a byte address, a register number is used to identify the to be the source or destination for an operand. All fixed length data types can fit in a register or pair of consecutive registers. The organization of the four data access types in the register set is shown below:

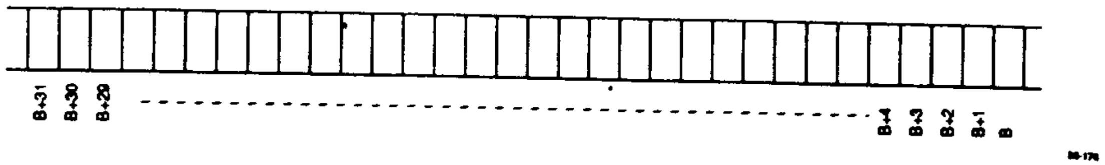


The lengths of the byte and halfword access types are shorter than the register length. These access types are right justified within the register. Only the lower portion of the register corresponding to the access type is significant and the upper portion will remain unaffected.

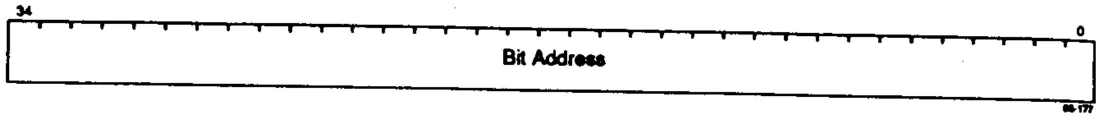
In the case of the doubleword access type, the operand occupies a pair of general purpose registers. The lower numbered register contains the least significant word while the higher numbered register contains the most significant word. However, since R31 cannot be used as the least significant register of a doubleword register pair, the results of using R31 as the source or destination operand of a doubleword access type is unpredictable.

Bit Addressing

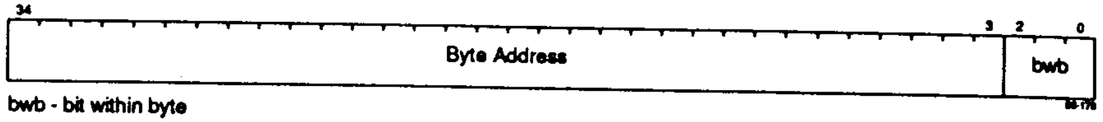
Bit addressing is employed to address data structures that are bit aligned, i.e., are aligned on an arbitrary bit boundary. However, unlike byte addressing which uses the byte as the atomic unit for memory addressing, bit addressing uses the bit as the basic addressing unit. Instead of using a four gigabyte virtual address space, bit addressing views the virtual address space as a thirty-two gigabit address space.



To address an arbitrary bit within a thirty-two gigabit address space requires a 35-bit address since there are 2^{32} bytes, each containing eight bits. A bit address is composed of two separate components, a 32-bit byte base address and a 32-bit bit offset. These components are combined to generate the 35-bit bit address.

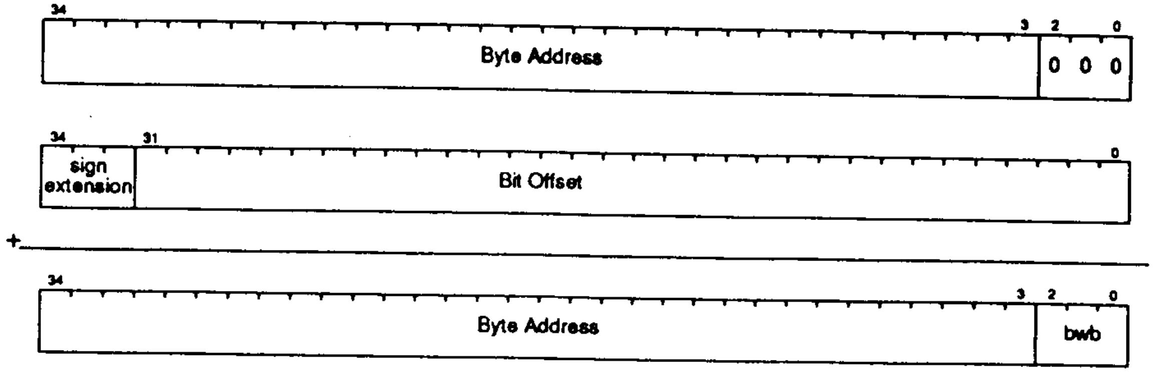


The byte address is zero extended on the right to 35-bit length. The **bwb** (bit within byte) field is initialized to zero to address the first bit (bit 0) within the byte. The sign extended 32-bit bit offset is added to form the bit address.



bwb - bit within byte

Once formed, the upper 32-bits of the bit address is used to identify the byte address of the operand with the lower three bits identifying the bit offset within the byte. The process of bit address generation is shown below:



Data Types

The µPD70616 recognizes a wide variety data types, typical of those utilized in most applications. Supported data types are listed below in Table 2.1.

Table 2.1. µPD70616 Data Types

Data Type		Length
Bit Data	bit	1–bit
Integer	byte halfword word doubleword	8–bits 16–bits 32–bits 64–bits
Unsigned Integer	byte halfword word doubleword	8–bits 16–bits 32–bits 64–bits
Floating Point	short real long real	32–bits 64–bits
Character String	byte string halfword string	8–bit characters 1 to 4 gigacharacters 16–bit characters 1 to 2 gigacharacters
Bit String		bit variable 1 to 4 gigabits
Bit Field		bit variable 0 to 32 bits

Signed Integers

Signed integers are expressed in two's complement binary notation. Four signed integer lengths are supported, byte, halfword, word and doubleword. Signed integer representation consists of a sign bit field and a magnitude field. The MSB (most significant bit) of a signed integer is the sign bit and indicates whether the number is positive or negative. The magnitude field contains the absolute value or magnitude of the signed integer.

Data Type	Range
Byte	–128 ~ 127
Halfword	–32768 ~ 32767
Word	–2147483648 ~ 2147483647
Doubleword	–9223372036854775808 ~ 9223372036854775807

Unsigned Integers

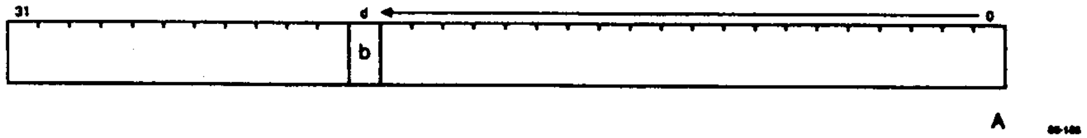
Unsigned integers represent natural numbers (non-negative integers) in binary notation. There are four unsigned integer data types: byte, halfword, word, and doubleword. Unsigned integers consist of only a magnitude field which is the same size as the data type. Unsigned integers are also used to represent the logical data types used for the bit-wise logical operations.

Data Type	Range
Byte	0 ~ 255
Halfword	0 ~ 65535
Word	0 ~ 4294967295
Doubleword	0 ~ 18446744073709551615

Bit

Bit data is often used to efficiently store data for control purposes. In the µPD70616, bit data is identified by a byte base address and a separate bit offset. The byte base address component selects the address of the word that contains the bit in question. The addressing mode is used to determine whether the operand is in a general purpose register (register addressing mode) or memory (all other addressing modes).

The bit offset is then used to identify the particular bit within the register/word that is to be manipulated. The bit offset is specified as a value in the range of [0..31]. Specifying any other value for the bit offset will cause an exception.



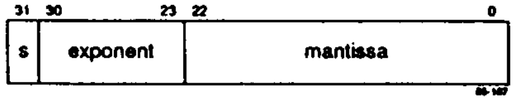
As a fixed length data type, bit data can reside in either a register or in memory. It is also possible to manipulate bit data using the bit string or bit field data types by specifying a length of 1 for these variable length data types.

Binary Floating Point

Binary floating point formats provide a wide range of numerical values over a specified level of precision. Floating point data types are useful for scientific and engineering calculations, numerical control, and any application requiring high performance numeric calculations such as graphics. The µPD70616 microprocessor supports basic floating point operations on two IEEE compatible binary floating point formats.

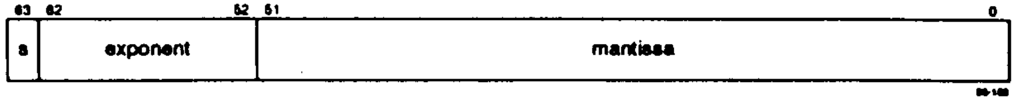
Short Real

The short real data type is a 32-bit binary floating point representation conforming to the IEEE single precision format. The short real format consists of a mantissa sign bit, an 8-bit biased exponent and a 23-bit mantissa as shown below:



Long Real

The long real data type is a 64-bit binary floating point representation conforming to the IEEE double precision format. The long real format consists of a mantissa sign bit, an 11-bit biased exponent and a 52-bit mantissa as shown below:



Decimal Data Type

The decimal data type is used for the manipulation of both packed and unpacked decimal numeric strings. The decimal data type divides each byte into two 4-bit fields (nibbles). In the packed decimal representation, each 4-bit field is assumed to contain a valid BCD (binary code decimal) digit in the range [0..9]. In the unpacked (or zoned) decimal representation, only the lower 4-bit field is assumed to contain a digit and the higher 4-bit bit field is called the zone field.

When a nibble is expected to contain a digit, only the valid BCD values [0..9] can be specified. Any other value will cause an illegal decimal format exception to occur. There is no restriction on the contents of the zone field.

Character Strings

Two types of character strings are recognized by the µPD70616 microprocessor, byte character strings and halfword character strings. Byte character strings are used to express standard ASCII text. Halfword character strings are also supported to express text needing a much wider range of characters than available with an 8-bit character set.

Instructions are provided for operating on character strings of either type and include:

- transfer
- comparison
- scanning
- skipping

Being a variable length data structure, a character string is fully defined by:

- the address of the start of the string
- the number of characters in the string

In addition to the above attributes, a character string must also obey the following restriction:

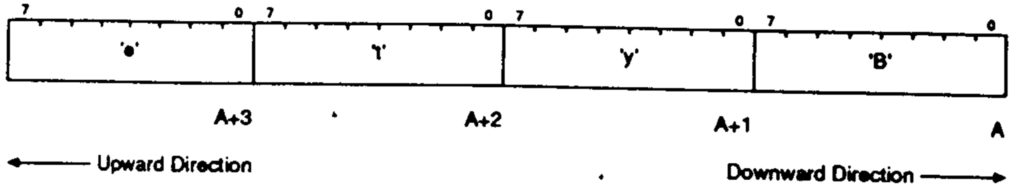
- the sum of the starting address and the length of a character string (in bytes) must be less than $2^{32} - 1$

Note that the number of characters and the length of a byte character string are the same while a halfword character has a byte length twice the number of characters in the string.

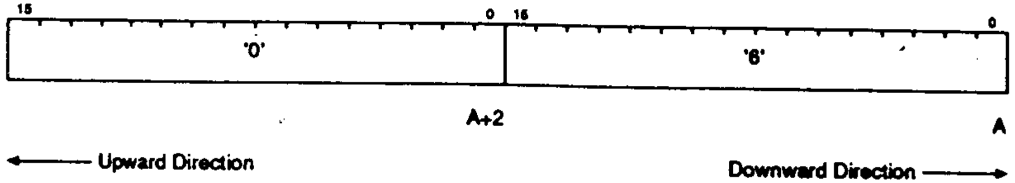
Some instruction permit specifying the direction of string processing. The direction within a character string in which addresses become larger is called the upward direction while the direction in which addresses become smaller is the downward direction. In all cases the ordering of characters within the string is in the upward (increasing addresses) direction. Only the direction of processing changes.

Examples of both byte and halfword character strings are shown below.

Byte character string (N=4)



Halfword character string (N=2)

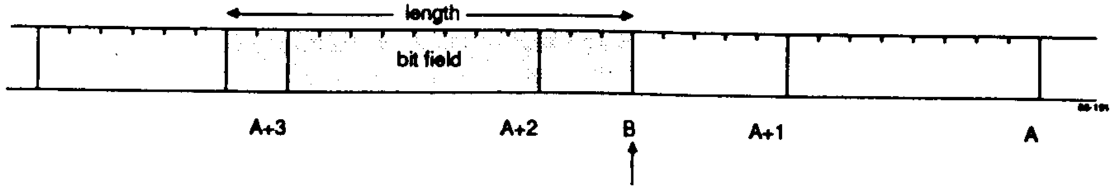


Bit Fields

Bit fields are a variable length data structure used to represent signed (two's complement notation) and unsigned integers in a compact format. An instance of the bit field data type can take any length between 0 and 32 bits, starting at any bit position in memory and subject only to the constraint that the bit field not span a length of greater than four bytes. As with the integer data types, bit 0 of a bit field is the least significant bit and in the case of signed bit fields, the most significant bit is the sign bit.

Being a variable length data structure, a bit field is fully defined by:

- the bit address (B) of the start of the bit field
- the length in bits



An integer x can be expressed in a bit field of length N if and only if

$$-2^{(N-1)} \le x < 2^{(N-1)} \quad \text{(signed)}$$
$$0 \le x < 2^{(N-1)} \quad \text{(unsigned)}$$

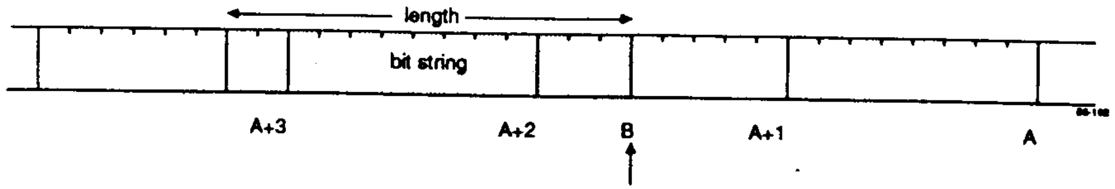
The integer value of a bit field of length 0 is zero.

Bit Strings

A bit string is a variable length logical data structure containing 0 to $2^{32} - 1$ bits. Applications of bit strings abound in applications as diverse as window management in bit-mapped graphic displays and for implementing set operations in high level languages.

Bit strings are treated as a logical data type with a full complement of monadic and dyadic operations defined. Being a variable length data structure, a bit string consists of the following two components:

- bit address (B) of the start of the bit string
- the length in bits



Like the character strings, the μPD70616 instruction set permits specifying the direction of bit string processing. The direction within a bit string in which addresses become larger is called the upward direction while the direction in which addresses become smaller is the downward direction.

Stacks

A stack is last-in-first-out (LIFO) data structure. The μPD70616 uses a push-down stack for a number of purposes including:

- subroutine return addresses
- saving program state during an interrupt or exception
- allocation of local variables during a procedure call

In the μPD70616, register R31 is the default stack pointer and is always assumed to be pointing to the current top of the stack (TOS). Before pushing new data on the stack, the stack is first decremented before copying the operand to the new TOS. In a similar manner, a stack pop operation will remove the current TOS and then increment the stack pointer.

The default stack pointer (R31) is assumed to a word (32-bit wide) stack. Any general purpose register can also be used to implement a stack by means of the autoincrement and autodecrement addressing modes.

Section 3
Register Set

The μPD70616 has a large number of general purpose and special purpose registers which are described in this section. The μPD70616 register set is divided into two categories. The **program register** set represents the set resources available to the application or user while the system programmer has in addition to the program register set the full resources of the **privileged register** set.

The program register set consists of the following 32-bit registers:

• general purpose registers	R0 – R28
• argument pointer (AP)	R29
• frame pointer (FP)	R30
• stack pointer (SP)	R31
• program counter	PC
• program status word (lower halfword)	PSW[15:0]

The privileged register set consists of the following additional registers (grouped by function):

Address Translation

• area table base registers 0 – 3	ATBR0 – ATBR3
• area table length registers 0 –3	ATLR0 – ATLR3

Stack Pointers

• level 0 – 3 stack pointers	L0SP – L3SP
• interrupt stack pointer	ISP

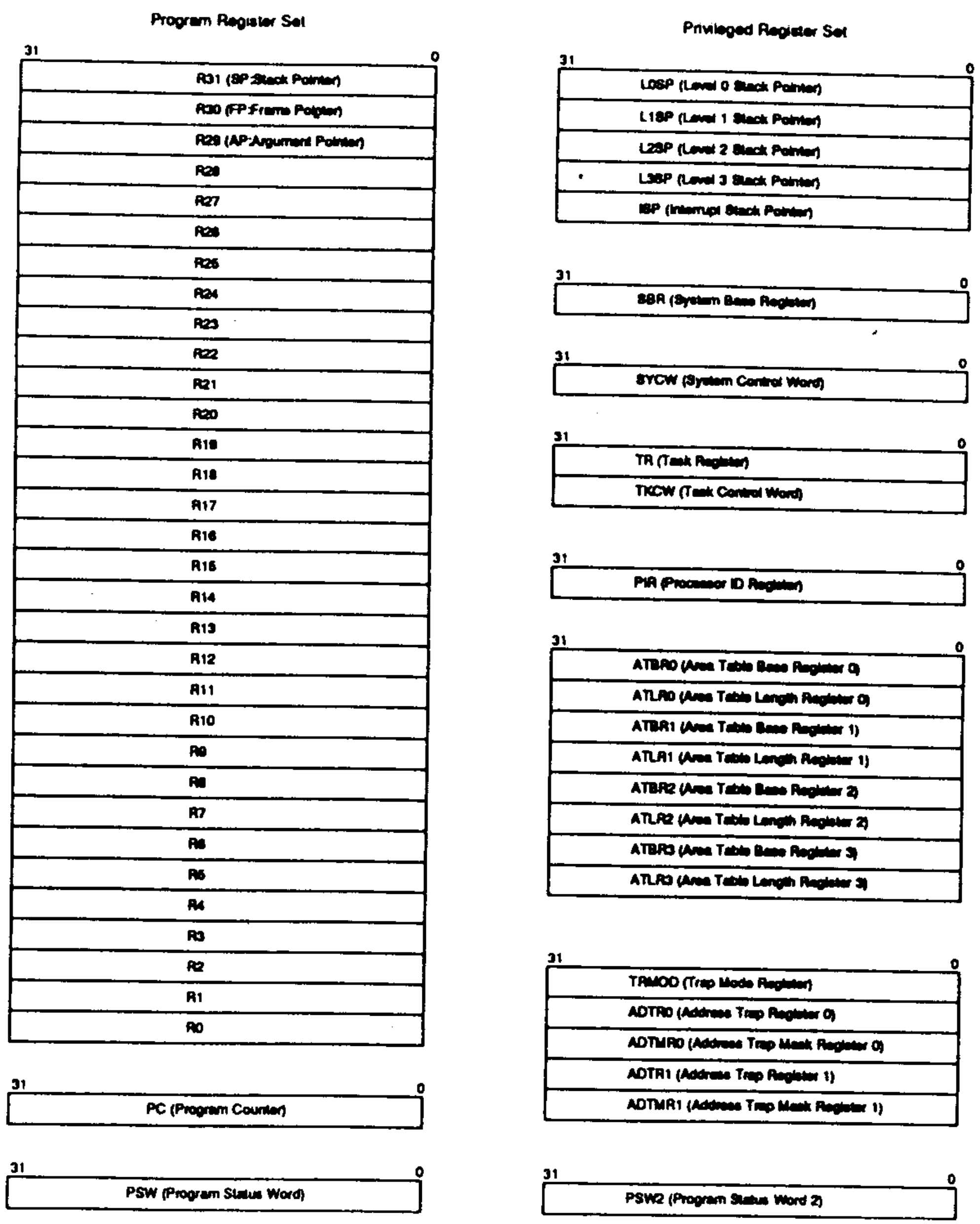
Debugging Registers

• trap mode register	TRMOD
• address trap registers	ADTR0 – ADTR1
• address trap mask registers	ADTMR0 – ADTMR1

Miscellaneous Registers

• program status word (upper halfword)	PSW[31:16]
• system base register	SBR
• system control word	SYCW
• task register	TR
• task control word	TKCW
• processor ID register	PID
• V20/V30 emulation mode PSW	PSW2

Figure 3–1. µPD70616 Register Set


Program Register Set
31
0
R31 (SP:Stack Pointer)
R30 (FP:Frame Poipter)
R29 (AP:Argument Pointer)
R28
R27
R26
R25
R24
R23
R22
R21
R20
R19
R18
R17
R16
R15
R14
R13
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
PC (Program Counter)
PSW (Program Status Word)
Privileged Register Set
L0SP (Level 0 Stack Pointer)
L1SP (Level 1 Stack Pointer)
L2SP (Level 2 Stack Pointer)
L3SP (Level 3 Stack Pointer)
ISP (Interrupt Stack Pointer)
SBR (System Base Register)
SYCW (System Control Word)
TR (Task Register)
TKCW (Task Control Word)
PIR (Processor ID Register)
ATBR0 (Area Table Base Register 0)
ATLR0 (Area Table Length Register 0)
ATBR1 (Area Table Base Register 1)
ATLR1 (Area Table Length Register 1)
ATBR2 (Area Table Base Register 2)
ATLR2 (Area Table Length Register 2)
ATBR3 (Area Table Base Register 3)
ATLR3 (Area Table Length Register 3)
TRMOD (Trap Mode Register)
ADTR0 (Address Trap Register 0)
ADTMR0 (Address Trap Mask Register 0)
ADTR1 (Address Trap Register 1)
ADTMR1 (Address Trap Mask Register 1)
PSW2 (Program Status Word 2)

Program Register Set

The program register set consists of the general purpose registers, the program counter (PC) and the program status word (PSW). Each of these registers is 32-bits wide and are available for use by application programs.

General Purpose Registers

The general purpose register set consists of thirty-two registers (R0 – R31) each 32-bits in width. A general purpose register can be used as an accumulator, base register, index register or to hold intermediate calculations. General purpose registers can be used at any execution level without restriction.

Three of the general purpose registers are reserved for specific purposes by certain instructions. R29 is called the **argument pointer** (AP) and is used to point to the list of procedure arguments by the CALL instruction. R30 is the **frame pointer** (FP) and is used to point to the current stack frame (work area for local variables and parameters) for currently executing procedure. R31 is the **stack pointer** (SP) and contains a pointer to the word on the current top stack (TOS).

The stack pointer is not a single register, but rather a cache of five registers separate stack pointers, one for each of the four execution levels and an interrupt stack pointer. The current execution level and external events such as interrupt and exceptions determine which of the five stack pointers is in use as the current stack pointer.

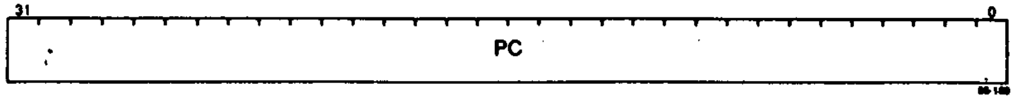
In addition to the AP, FP, and SP registers, other general purpose registers are required by string instructions to allow the instruction to resumed following an interrupt or exception. In this case, registers are reserved for use starting from R28 and allocated in a downward direction.

Program Counter

The program counter (PC) is a register which contains the memory address of the first byte of the instruction currently being executed. The PC contains a virtual address in the virtual address mode and a physical address in the real address mode.

The PC is a 32-bit register which cannot be directly read or written. However, the contents of the PC can be examined by the instruction

```
movea    0[ pc ], dest          -- compute the effective address using the
                                -- pc with no displacement
```



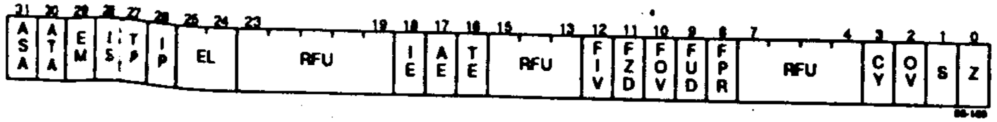
bits 31:0 PC The address of the first byte of the currently executing instruction. The PC contains a virtual address in the virtual address mode, a physical address in the real address mode.

Program Status Word

The program status word (PSW) is a 32-bit register containing program status and control information. The PSW is divided into upper and lower halfwords with the upper halfword being modified only by means of the privileged UPDPSW.W instruction. The lower halfword of the PSW has two fields containing the integer and floating point condition codes. The upper halfword contains the processor control and status fields for the currently executing task.

The contents of the PSW can be read regardless of the execution level. The PSW is modified according to the following conditions:

- the integer and floating point condition codes can be modified using the UPDPSW.H instruction
- the control and condition code fields can be modified at execution level 0 by the privileged UPDPSW.W instruction
- the status field is modified by the execution of certain instructions such as CHLVL and RETIS



bit 0 Z

The Z (zero) flag indicates if the results of the operation were zero.

Z = 0 result is non-zero
Z = 1 result is zero

bit 1 S

The S (sign) flag indicates if the results are negative (signed) or if the MSB is set (unsigned).

S = 0 result is positive or zero or MSB is 0
S = 1 result is negative or MSB is set

bit 2 OV

The OV (overflow) flag indicates if an overflow occurred.

OV = 0 no overflow
OV = 1 overflow

bit 3 CY

The CY (carry) flag indicates if a carry or borrow was generated as a result of the operation.

CY = 0 no carry (borrow) generated
CY = 1 carry (borrow) was generated

bits 4:7 RFU

Reserved for future use

bit 8 FPR

The FPR (precision) flag indicates the exactness of a floating point operation.

FPR = 0 exact result
FPR = 1 inexact result

bit 9 FUD

The FUD (underflow) flag indicates if an underflow occurred as the result of a floating point operation.

FUD = 0 no floating point underflow
FUD = 1 floating point underflow occurred

bit 10 FOV

The FOV (overflow) flag indicates whether an overflow occurred as a result of a floating point operation.

FOV = 0 no floating point overflow
FOV = 1 floating point overflow occurred

bit 11 FZD

The FZD (zero divide) flag indicates if a zero division took place.

FZD = 0 no floating point zero divide
FZD = 1 a floating point zero divide occurred

bit 12 FIV

The FIV (invalid) flag indicates the occurrence of an invalid floating point operation.

FIV = 0 no invalid operation occurred
FIV = 1 invalid operation occurred

bits 13:15 RFU

Reserved for future use

bit 16 TE

The TE (trace enable) flag enables and disables instruction trace.

TE = 0 instruction trace disabled
TE = 1 instruction trace enabled

bit 17 AE

The AE (address trap enable) flag is a global enable/disable for the address trap logic.

AE = 0 address traps disabled
AE = 1 address traps enabled

bit 18 IE

The IE (interrupt enable) flag permits software to selectively enable and disable maskable interrupts.

IE = 0 maskable interrupts disabled
IE = 1 maskable interrupts enabled

bits 19:23 RFU

Reserved for future use

bits 24:25 EL

The EL (execution level) field contains the value of the current execution level.

EL = 00 execution level 0 (privileged)
EL = 01 execution level 1
EL = 10 execution level 2
EL = 11 execution level 3

bit 26 IP

The IP (instruction pending) flag indicates whether or not an instruction has been interrupted and should be resumed.

IP = 0 no instruction pending
IP = 1 instruction pending

bit 27 TP

The TP (trace pending) flag controls the guarantees the occurance of a single instruction trace for each instruction.

TP = 0 instruction trace not pending
TP = 1 instruction trace pending

bit 28 IS

The IS (interrupt stack) flag indicates whether the current processor context is in the interrupt space.

IS = 0 interrupt stack inactive
IS = 1 interrupt stack active

bit 29 EM The EM (emulation mode) flag indicates the current processor mode.

EM = 0	native mode
EM = 1	emulation mode

bit 30 ATA The ATA (asynchronous task trap active) flag indicate whether an asynchronous task trap processing is in progress.

ATA = 0	ATT processing not in progress
ATA = 1	ATT processing in progress

bit 31 ASA The ASA (asynchronous system trap active) flag indicate whether an asynchronous system trap processing is in progress.

ASA = 0	AST processing not in progress
ASA = 1	AST processing in progress

Privileged Register Set

Twenty-two 32-bit registers make up the µPD70616 privileged register set. The privileged register set controls the execution environment and are accessible only to programs executing at privilege level 0.

The contents of privileged registers are examined and modified using the load privileged register (LDPR) and store privileged register (STPR) instructions. Each privileged register is assigned a register ID number (figure 3–2) which is used to identify the source or destination privileged register.

Figure 3–2. Privileged Register ID Numbers

ID	Privileged Register		Permissible Operations	
			LDPR	STPR
0	ISP	Interrupt Stack Pointer	O	O
1	L0SP	Level 0 Stack Pointer	O	O
2	L1SP	Level 1 Stack Pointer	O	O
3	L2SP	Level 2 Stack Pointer	O	O
4	L3SP	Level 3 Stack Pointer	O	O
5	SBR	System Base Register	O	O
6	TR	Task Register	–	O
7	SYCW	System Control Word	O	O
8	TKCW	Task Control Word	O	O
9	PIR	Processor ID Register	–	O
10–14		Reserved for future use	X	X
15	PSW2	Emulation Mode Program Status Word	O	O
16	ATBR0	Area Table Base Register 0	O	O
17	ATLR0	Area Table Length Register 0	O	O
18	ATBR1	Area Table Base Register 1	O	O
19	ATLR1	Area Table Length Register 1	O	O
20	ATBR2	Area Table Base Register 2	O	O
21	ATLR2	Area Table Length Register 2	O	O
22	ATBR3	Area Table Base Register 3	O	O
23	ATLR3	Area Table Length Register 3	O	O
24	TRMOD	Trap Mode Register	O	O
25	ADTR0	Address Trap Register 0	O	O
26	ADTR1	Address Trap Register 1	O	O
27	ADTMR0	Address Trap Mask Register 0	O	O
28	ADTMR1	Address Trap Mask Register 1	O	O
29–31		Reserved for future use	X	X

85-050

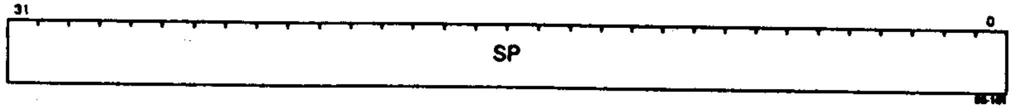
In addition to the LDPR and STPR instructions, other privileged instructions may also implicitly select a privileged register as an instruction operand.

Stack Pointers

The stack pointer (R31) is part of the program register set and contains the virtual address of the current top of stack. In actuality, there is a cache of five independent stack pointers, one for each of the four execution levels and a separate interrupt stack pointer. The stack pointer selected when R31 is referenced by an instruction is determined by the value of the PSW IS and EL fields as defined below:

IS	EL	Selected Stack Pointer
0	00	level 0 stack pointer
0	01	level 1 stack pointer
0	10	level 2 stack pointer
0	11	level 3 stack pointer
1	00	interrupt stack pointer
1	01	undefined
1	10	undefined
1	11	undefined

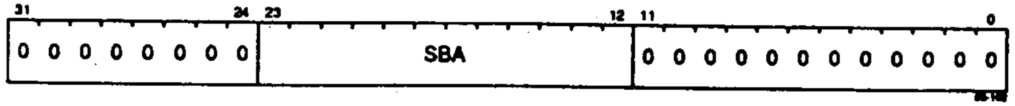
Programs at execution level 0 can access any of the stack pointers by means of the LDPR and STPR instructions. Load and store operations to R31 only affect the stack pointer for the current execution level.



bits 0:31 SP The SP (stack pointer) contains the 32-bit virtual address to the current top of stack.

System Base Register

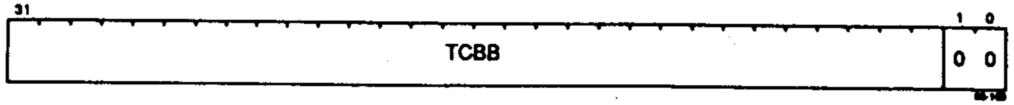
The System Base Register (SBR) contains the pointer to the first entry of the system base table. The system base table is a table of vectors used for interrupt and exception processing. The system base register contains the physical address of the start of this table which must be aligned on a page boundary.



bits 0:31 SBA The SBA (system base address) is the physical address of the system base table. The system base table is aligned on a page boundary and the twelve low order bits (bits 0:11) must be zero, otherwise the results are UNPREDICTABLE. The upper eight bits (bits 24:31) are unused by the µPD70616 but should zero for compatibility with future microprocessors.

Task Register

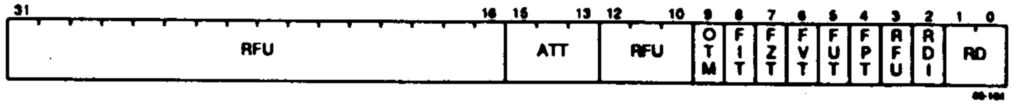
The Task Register (TR) holds the virtual address of the task control block (TCB) for the currently installed context. The Task Register is a 32-bit read-only register and is loaded automatically by the load task context instruction.



bits 0:31 TCBB The TCBB (Task Control Block Base) is the virtual address of the TCB for the current context. A task control block must be aligned on a word boundary, thus TCBB field (bits 0:1) must be zero, otherwise the results are UNPREDICTABLE.

Task Control Word

The Task Control Word (TKCW) contains task specific information and is swapped in and out out as part of the task context.



bits 0:1 RD The RD (float rounding control) field controls the direction of rounding during floating point calculations.

RD = 00	round toward nearest
RD = 01	round toward −∞
RD = 10	round toward +∞
RD = 11	round toward zero

bit 2 RDI The RDI (integer rounding control) field controls the direction of rounding during floating point to integer conversions.

RDI = 0	use RD field rounding mode
RDI = 1	round toward zero

bit 3 RFU Reserved for future use

bit 4 FPT The FPT (floating point precision trap enable) flag selects the desired mode for the handling of precision errors.

FPT = 0	floating point precision traps disabled
FPT = 1	floating point precision traps enabled

bit 5 FUT The FUT (floating point underflow trap enable) flag selects the desired mode for the handling a floating point underflow condition.

FUT = 0	floating point underflow traps disabled
FUT = 1	floating point underflow traps enabled

bit 6 FVT The FVT (floating point overflow trap enable) flag selects the desired mode for the handling of overflow errors.

FVT = 0 floating point overflow traps disabled
FVT = 1 floating point overflow traps enabled

bit 7 FZT The FZT (floating point zero divide trap enable) flag selects the desired mode for the handling of a zero floating point divisor.

FZT = 0 floating point zero divide traps disabled
FZT = 1 floating point zero divide traps enabled

bit 8 FIT The FIT (floating point invalid operation trap enable) flag selects the desired mode for the handling an invalid floating point operation.

FIT = 0 invalid floating point operation traps disabled
FIT = 1 invalid floating point operation traps enabled

bit 9 OTM The OTM (operand trap mask) flag controls the handling of reserved floating point operands such as infinities or NaNs.

OTM = 0 trapping enabled
OTM = 1 trapping disabled

This flag has a different than other flags associated with floating point operation traps. This field is cleared to 0 in the µPD70616 microprocessor.

bits 10:12 RFU Reserved for future use

bits 13:15 ATT The ATT (asynchronous task trap level) field contains the value of the execution level required to trigger an asynchronous task trap (ATT).

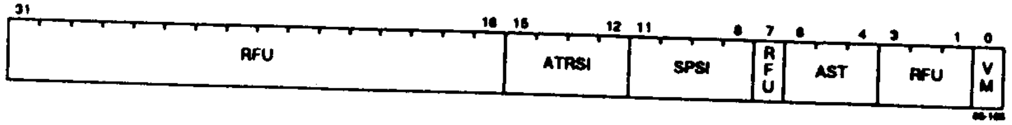
ATT[15:13] =

0	0	0	level 0 ATT
0	0	1	level 1 ATT
0	1	0	level 2 ATT
0	1	1	level 3 ATT
1	0	0	ATT disabled
1	0	1	ATT disabled
1	1	0	ATT disabled
1	1	1	ATT disabled

bits 16:31 RFU Reserved for future use

System Control Word

The System Control Word (SYCW) is the system-wide register for the control of the operating environment.



bit 0 VM The VM (virtual mode) field controls the operating mode of the processor.

VM = 0 physical address mode
VM = 1 virtual address mode

bits 1:3 RFU Reserved for future use

bits 4:6 AST The AST (asynchronous system trap level) field contains the value of the execution level required to trigger an asynchronous system trap (AST).

AST[6:4] =

0	0	0	level 0 AST
0	0	1	level 1 AST
0	1	0	level 2 AST
0	1	1	level 3 AST
1	0	0	AST disabled
1	0	1	AST disabled
1	1	0	AST disabled
1	1	1	AST disabled

bits 8:11 SPSI The SPSI (stack point switching inhibited) field controls the change of the stack pointers during context switching.

SPSI[8] =
0 level 0 stack pointer is fixed
1 level 0 stack pointer is switched

SPSI[9] =
0 level 1 stack pointer is fixed
1 level 1 stack pointer is switched

SPSI[10] =
0 level 2 stack pointer is fixed
1 level 2 stack pointer is switched

SPSI[11] =
0 level 3 stack pointer is fixed
1 level 3 stack pointer is switched

bits 12:15 ATRSI The ATRSI (area table register switching inhibited) field controls the change of the area table registers during context switching.

ATRSI[12] =
- 0 the section 00 area table registers are fixed
- 1 the section 00 area table registers are switched

ATRSI[13] =
- 0 the section 01 area table registers are fixed
- 1 the section 01 area table registers are switched

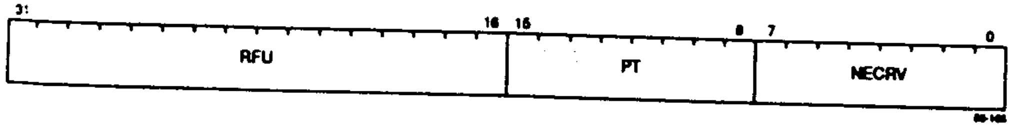
ATRSI[14] =
- 0 the section 10 area table registers are fixed
- 1 the section 10 area table registers are switched

ATRSI[15] =
- 0 the section 11 area table registers are fixed
- 1 the section 11 area table registers are switched

bits 16:31 RFU Reserved for future use

Processor ID Register

The Processor ID Register (PIR) is a read-only register containing identification information about the processor.

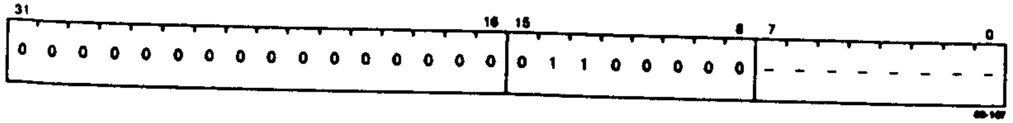


bits 0:7 NECRV This field is reserved for use by NEC.

bits 8:15 PT The PT (processor type) field contains information identifying the CPU.

bits 16:31 RFU Reserved for future use

The contents of the PIR register for the µPD70616 (V60) microprocessor is

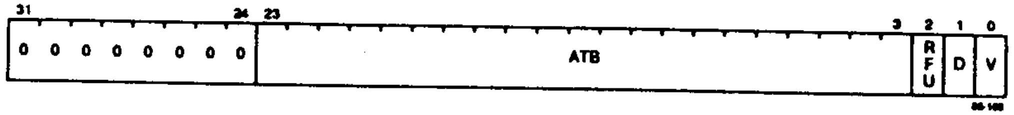


Area Table Registers

The µPD70616 includes a on-chip demand paged memory management unit for applications requiring a large virtual address space (refer to section 4 for detailed information). The virtual address is divided into four one gigabyte sections. Each section has a pair of registers defining the location and size of the associated area tables. Since the area table base (ATBR0–ATBR3) registers and area table length registers (ATLR0–ATLR3) are used as pair, they are commonly referred to as area table registers (ATBR/ATLR).

Area Table Base Registers

The four Area Table Base Registers (ATBR0–ATBR3) contain the base address and control information for managing the four area tables.



bit 0 V The V (valid) field determines if the contents of the area table registers are valid. An exception will occur if an address translation is attempted using an invalid ATBR/ATLR pair.

bit 1 D The D (direction) field determines the direction of growth for this section.

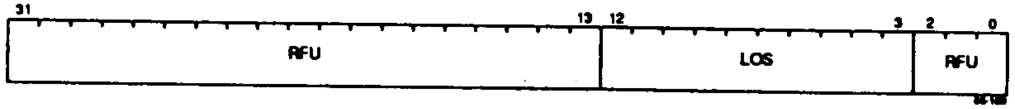
D = 0 growth direction is positive (increasing addresses)
D = 1 growth direction is negative (decreasing addresses)

bit 2 RFU Reserved for future use

bits 3:31 ATB The ATB (area table base address) is a 29-bit physical address of the first area table entry for the associated section. The lower order three address bits (bits 0:2) are zero and the area table must be aligned on a doubleword boundary. The high order eight bits (bits 24:31) are ignored by the µPD70616 microprocessor but must be zero for compatibility with future versions.

Area Table Length Registers

The four Area Table Length Registers (ATLR0–ATLR3) contain information about the size of the associated area table as shown below.



bits 0:2 RFU Reserved for future use

bits 3:12 LOS The LOS (limit of section) field is used by the hardware to determine the number of valid area table entries in an area table. If the D bit is cleared (positive growth direction), area table entries from 0 ≤ n ≤ LOS are considered valid. If the D bit is set (negative growth direction), area table entries in the range LOS ≤ n ≤ 1023 are valid.

bits 13:31 RFU Reserved for future use

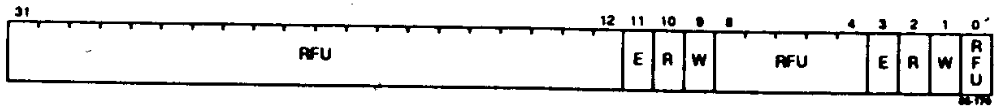
Address Trap Registers

The five address trap registers provide a powerful tool for the debugging and verification of software. These registers can be programmed so that an exception (refer to section 9 for details) occurs when a matching selected and access type is generated. The five address trap registers are:

- trap mode register
- trap address registers
- trap address mask registers

Trap Mode Register

The trap mode register specifies what kinds of memory access (read, write and execute), if any, are to generate traps.



bit 0 RFU Reserved for future use

bit 1 W The W (write) field controls the trapping of a write access into the region defined by the ADTR0/ADTMR0 registers.

W = 0 ADTR0 write access traps disabled
W = 1 ADTR0 write access traps enabled

bit 2 R The R (read) field controls the trapping of a write access into the region defined by the ADTR0/ADTMR0 registers.

R = 0 ADTR0 read access traps disabled
R = 1 ADTR0 read access traps enabled

bit 3 E The E (execute) field controls the trapping of a write access into the region defined by the ADTR0/ADTMR0 registers.

E = 0 ADTR0 execute access traps disabled
E = 1 ADTR0 execute access traps enabled

bits 4:8 RFU Reserved for future use

bit 9 W The W (write) field controls the trapping of a write access into the region defined by the ADTR1/ADTMR1 registers.

W = 0 ADTR1 write access traps disabled
W = 1 ADTR1 write access traps enabled

bit 10 R The R (read) field controls the trapping of a write access into the region defined by the ADTR1/ADTMR1 registers.

R = 0 ADTR1 read access traps disabled
R = 1 ADTR1 read access traps enabled

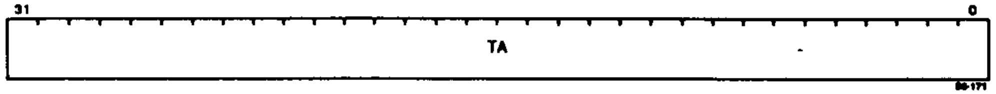
bit 11 E The E (execute) field controls the trapping of a write access into the region defined by the ADTR1/ADTMR1 registers.

E = 0 ADTR1 execute access traps disabled
E = 1 ADTR1 execute access traps enabled

bits 12:31 RFU Reserved for future use

Address Mask Registers

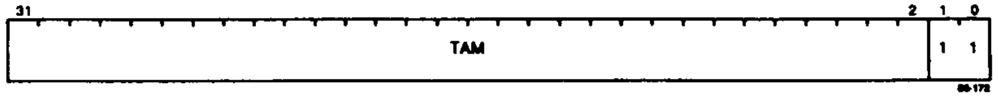
The Address Trap Registers (ADTR0/ADTR1) are used to specify the base address of the range of addresses to be trapped.



bits 0:31 TA The TA (trap address) field holds the target virtual address to cause a trap.

Address Trap Mask Registers

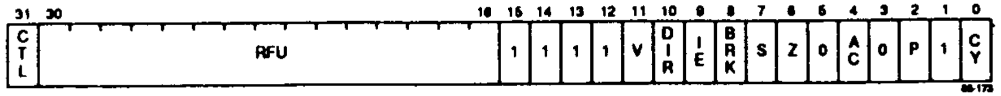
The Address Trap Mask Registers (ADTMR0/ADTMR1) are used to mark address bits in the associated ADTR register as "don't care", allowing accesses in a range of addresses to be trapped.



bit 0:31 TAM The TAM (trap address mask) field contains the mask pattern used to specify a range of trap addresses. Note that the lower two bits of the TAM field (bits 0:1) must be set, otherwise the results are UNPREDICTABLE.

Program Status Word 2

Program Status Word 2 (PSW2) functions as the V20/V30 emulation mode program status word. Refer to section 10 for details on the operation of this register.



Section 4
Address Spaces

This section describes the configuration and operation of the different µPD70616 address spaces. The µPD70616 microprocessor provides three separate address spaces:

- a sixteen megabyte (16MB) memory address space
- a sixteen megabyte (16MB) I/O address space
- a four gigabyte (4GB) virtual address space

The implementation of the four gigabyte virtual address space is supported by an on-chip memory management unit (MMU). Memory management is utilized to give each program or task the illusion of exclusive access to a four gigabyte (4 billion byte) linear address space. A virtual address space is mapped onto the memory address space (physical memory addresses) or I/O address space (peripheral device addresses) by the MMU in a process known as address translation. Address translation is a two-level process where a virtual address is converted to a physical address (in either the memory or I/O spaces) and all associated permissions are verified prior to the actual execution of the bus cycle.

Introduction

The memory address space uses the physical addresses of memory devices and limits the size to the maximum number of addressable bytes of memory. Since the µPD70616 microprocessor has twenty-four external address lines, the memory address space ranges from physical address 0 (0x000000) to an upper address of 224 – 1 (0xFFFFFF) in units of one byte. Because most systems do not implement a full sixteen megabyte memory address space since the actual size of the physical memory sub-system is determined by the application. There are no requirements for the alignment of data within the memory address space.

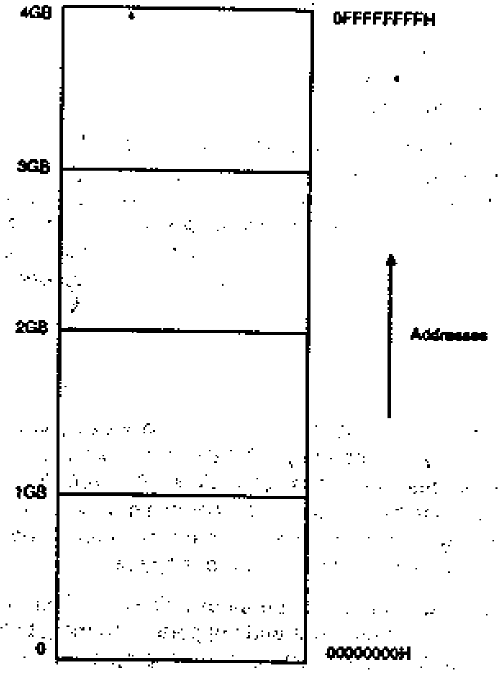
The I/O address space is used to access and control peripheral devices. Like the memory address space, the I/O address space uses all of the µPD70616 address lines and supports a 16 megabyte address space. The I/O address space is accessed by the execution of I/O instructions or by mapping a portion of the virtual address into the I/O address space.

The virtual address space is a much larger address space using 32-bit virtual addresses. By the process of address translation, the virtual address space is mapped onto one of the physical address spaces by the memory management unit. To aid in the task of managing such a large address space, the virtual address space is paged using a four kilobyte (4KB) page size.

Virtual Address Space

The virtual address space is viewed by programs as a byte addressable four gigabyte (2^{32} bytes) linear address space.

Figure 4-1. Virtual Address Space

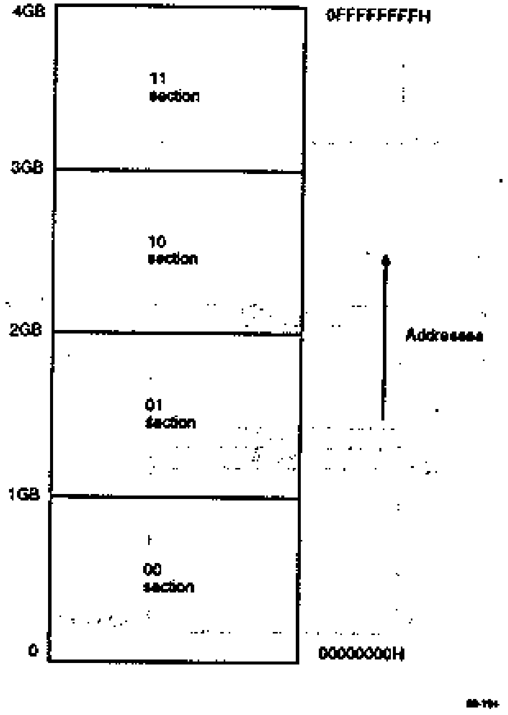


Sections

The four gigabyte virtual address space is divided into four one gigabyte (1GB) sections. The lowest address section is the 00 section and the remaining sections are called the 01, 10, and 11 sections respectively.

In the µPD70616 microprocessor, a section is a unit of sharing for a common virtual space in a system employing multiple virtual spaces.

Figure 4-2. Sections

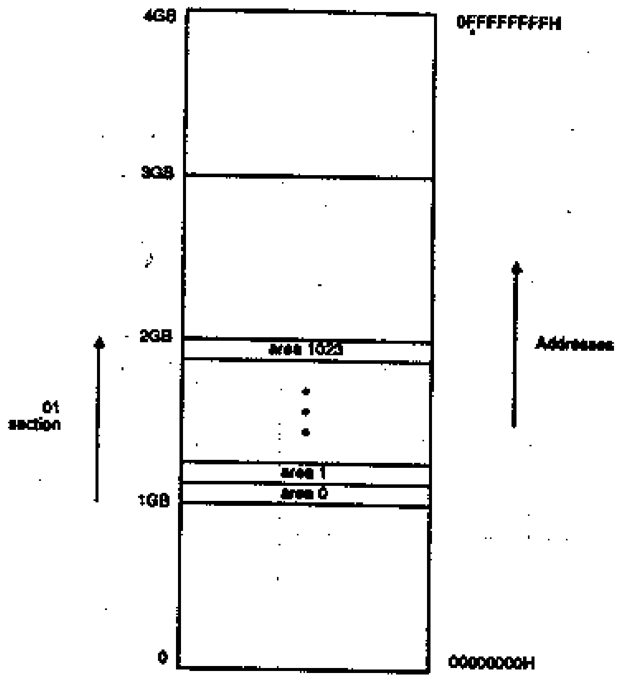


Areas

Each section is divided into 1024 areas. Each area is one megabyte (1MB) in size. The lowest numbered area within a section is area 0 and numbering proceeds up to the last area, area 1023.

The area permits protection of system resources and private sharing of virtual address space at the task level.

Figure 4-3. Areas

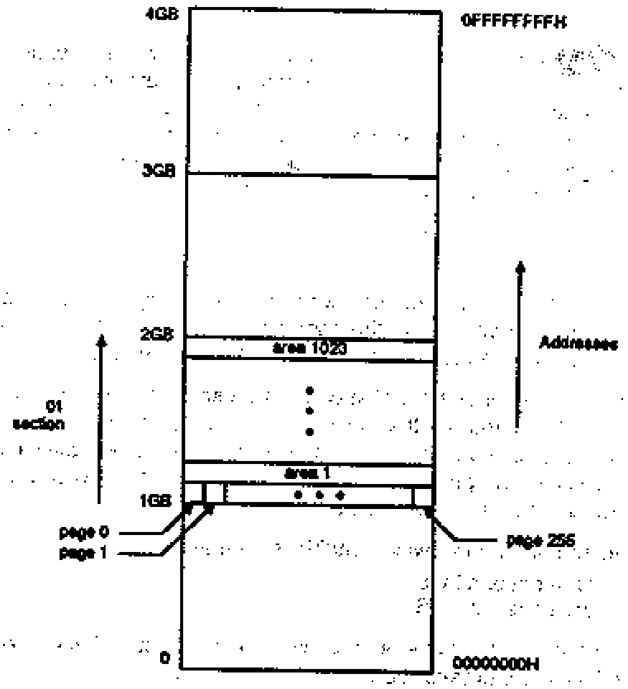


Pages

Each area consists of 256 pages, each page four kilobytes (4KB) in size. Pages are numbered from page 0 in ascending order to page 255 within an area.

The page is the smallest unit of virtual memory management. Pages are also the unit of mapping a portion of the virtual space into the I/O address space.

Figure 4-4. Pages



Sections

A section is a 1GB segment of the virtual address space and is the basic memory management unit for a system having multiple virtual spaces. Consider the following virtual memory configuration:

- section 11................operating system
- section 10................common system and utilities
- section 01.................task control information
- section 00.................task work region

In this example, all tasks share the virtual space defined by section 10 and 11 and have private virtual spaces defined by sections 00 and 01.

The µPD70616 microprocessor places no restrictions on the assignment of virtual spaces to tasks. Each task can have a unique virtual space or multiple tasks can share a common virtual address space. When several tasks share a virtual space, the virtual space (defined by one or more area table register pairs) remains static while the tasks are swapped in and out of the processor. Virtual spaces are switched only when the virtual spaces differ between tasks. Because the operating system is shared by all tasks, it can reside in a fixed section. Fixing of a section is selected by the ATRSI fields in the SYCW (System Control Word).

Section Length

A section need not occupy the entire 1 GB virtual address space and is usually much smaller. The µPD70616 can limit the size of a section such that the number and size of the address translation tables is minimized. Setting the size of a section requires specifying a length of section (LOS) value in the area table register.

The LOS field selects the maximum number of areas that comprise a section. Along with the D (direction) bit, the LOS field determines which areas are defined and whether growth is from the bottom or top of the section. When the D bit is cleared, the growth direction is positive and areas are measured from the base of the section (area 0) and continuing to area LOS–1. When the D bit is set, the growth direction is negative and areas are allocated from the top of the section (area 1023) and continue down to area 1023–LOS.

Legitimate area table references require the following conditions be met:

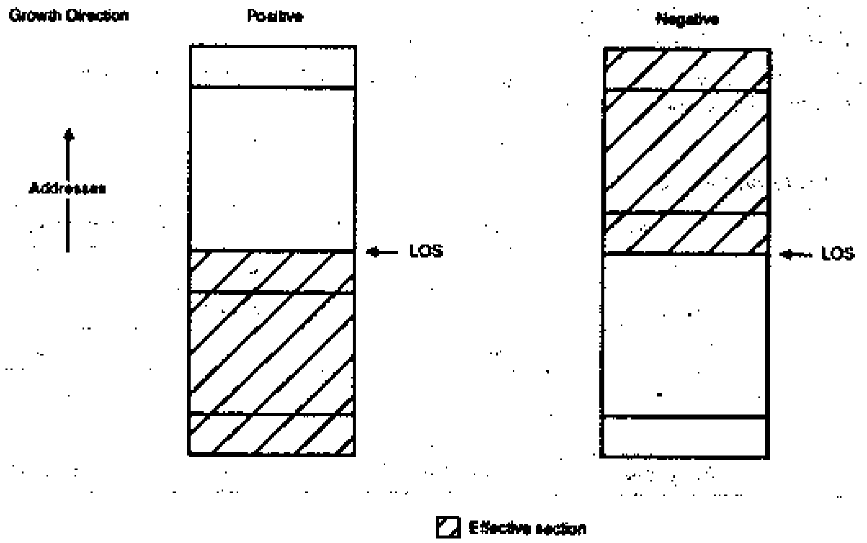
• positive growth	area offset ≤ LOS
• negative growth	area offset ≥ LOS

An attempt to reference an area outside the these regions will cause a Section Length Violation exception. Refer to figure 4–5 for details of section limits and organization.

Table 4–1. Section Limits and Ranges

LOS	Positive Growth Direction		Negative Growth Direction	
	Areas	Range	Areas	Range
LOS = 0	1 area	area 0	1024 areas	areas 1023 – 0
LOS = 1	2 areas	areas 0 – 1	1023 areas	areas 1023 – 1
...	...	...	...	...
...	...	...	...	...
LOS = 1023	1024 areas	areas 0 – 1023	1 area	area 1023

Figure 4–5. Section Limits and Ranges



Undefined Sections

A section which has not been defined is called an undefined section. A section is considered to be undefined if the V (valid) bit in the associated area table register is cleared. An invalid Section exception will be raised if an attempt to access an undefined section is made.

Areas

An area is a 1MB segment of the virtual address space and is available for private sharing of a virtual space such as between two tasks. An area can be shared by simply copying the area table entry.

The area is also a unit of protection between tasks. Both execution level and access types are checked prior to a references to an area. Swapping of entire areas is also supported.

Area Length

An area need not occupy the entire 1 MB virtual address space. The µPD70616 can limit the size of an area such that the number and size of the page tables is minimized. Setting the size of a area requires specifying a length of area (LOA) value in the area table entry (ATE).

The LOA field selects the maximum number of pages that comprise an area. Along with the ATE D (direction) bit, the LOA field determines which pages are defined and whether growth is from the bottom or top of the area. When the D bit is cleared, the growth direction is positive and pages are allocated from the base of the area (page 0) and continuing to page LOA–1. When the D bit is set, the growth direction is negative and pages are measured from the top of the area (page 255) and continuing down to area 255–LOA.

Legitimate page table references require the following conditions be met:

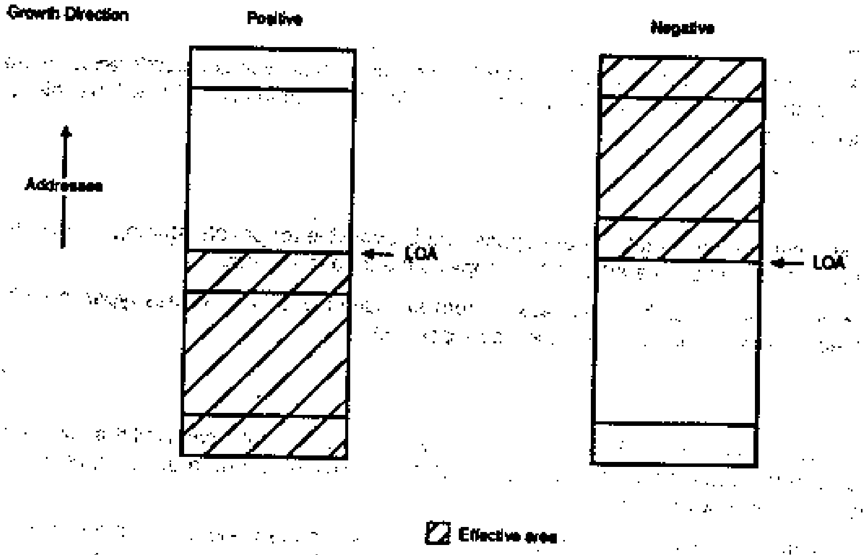
- positive growth page offset ≤ LOA
- negative growth page offset ≥ LOA

An attempt to reference an page outside these regions will cause a Area Length Violation exception. Refer to figure 4–6 for details of area limits and organization.

Table 4–2. Area Limits and Ranges

LOA	Positive Growth Direction		Negative Growth Direction	
	Pages	Range	Pages	Range
LOA = 0	1 page	page 0	256 pages	pages 255 – 0
LOA = 1	2 pages	pages 0 – 1	255 pages	pages 255 – 1
...	...	...	...	...
...	...	...	...	...
LOA = 255	256 pages	pages 0 – 255	1 page	page 255

Figure 4–6. Area Limits and Ranges



Protection

Protection and separation of tasks from each other and the operating system is one of the primary advantages of a virtual memory system. Protection mechanisms enhance the operation of the system by detecting and preventing attempts by tasks to access programs and data without first having established the necessary permissions. The µPD70616 protection mechanisms are implemented using three levels. First an execution level is established which determines the relative trustworthiness of a task. A check of execution level and access type permission are performed at the area level while an independent access type permission check is performed again at the page level.

Execution Levels

The execution level of a task determines whether a task is privileged or non-privileged. The execution level (EL) is a 2-bit field in the PSW register which contains the current execution level. Execution levels are numbered from 0 to 3 with level 0 being the most privileged and level 3 being the least privileged. Programs execution at level 0 are said to be privileged and can execute the entire µPD70616 instruction set. Programs executing at other execution levels (levels 1, 2 and 3) are are said to be non-privileged and attempts to execute a privileged instruction will cause an exception. In a typical system, the operating system kernel is executed at level 0 while tasks and non-privileged operating system utilities are executed at levels 1 to 3.

Area Protection

Protection for an area (1MB region) is defined by the corresponding entry in the area table. Area level protection is specified independently for each access method (Read, Write and Execute) by a 2-bit field. Each of these protection three fields contain the minimum execution level necessary to allow access to the area. For example, if

Read level..............2
Write level..............0
Execute level.........1

are specified, then access to the area by program execution level is:

level 0	Read, Write and Execute
level 1	Read and Execute
level 2	Read
level 3	no access

The area protection mechanisms provide a flexible means to restrict access to an area by execution level and access type.

Page Protections

Page protection defines the permissions required to access a page (4KB) and are specified in the corresponding page table entry. Page protections permit or deny access on the basis of contents of the page. For example, if the page protections are

Read	true
Write	true
Execute	false

then the contents of the page are readable and writable but not executable.

An access is possible only when both area and page permissions have been granted. Any attempt at access without both sets of permissions will cause an exception. Protection mechanisms are enabled only when the processor is operating in the virtual mode. In the physical address mode, all protection mechanisms are disabled and no exceptions occur.

Memory Address Space

The memory address space is the space all physical memory accesses occur in and defines the maximum amount of addressable memory. Because of the restrictions imposed on the memory address space by the processor model and and system requirements, the memory address space will vary from system to system. The µPD70616 microprocessor has a memory address space of sixteen megabytes (16MB) starting from address 0x000000 and proceeding linearly to an upper address of 0xFFFFFF in units of a single byte.

The µPD70616 can access the memory address space from both the virtual and physical modes. In the physical address mode, a logical address and physical address are the same and are output without modification. In virtual mode, the virtual address is first translated to a physical address before being output. The selection of virtual/physical addresses is done by the SYCW register (refer to section 3 for details).

Previously, all descriptions have assumed operation in the virtual address mode. When operated in the physical address mode the following address space differences should be noted:

- only one address space
- no protection
- address size is restricted
- system/processor dependent design

The µPD70616 instruction set is fully functional in the physical address mode. However, because a physical memory address is 24-bits, the upper eight bits of an address (bits 24:31) must be zero (MBZ). The µPD70616 will tolerate the upper order eight bits becoming non-zero during effective address calculation but the results may be unpredictable.

I/O Address Space

The I/O address space is used for the placement and control of peripheral devices without requiring the reservation of a portion of the memory address space. Like the memory address space, the I/O address space is 16MB in size with addresses ranging from 0x000000 to 0xFFFFFFFF in byte units. The valid I/O address space access ranges are completely determined by each individual system.

Two different methods are provided to generate I/O space accesses to maximize system software flexibility. The first approach uses the traditional method of dedicated I/O instructions to generate I/O address space accesses. The alternate approach uses the memory management and address translation logic to map a portion of the virtual space into the I/O address space.

I/O Space Access

I/O space accesses are always generated by the execution of the privileged IN and OUT instructions. An I/O port address is specified as a 32-bit operand but because of the external address bus size restriction, bits 24:31 of a port address must be zero. The operation using an I/O port address outside the range 0x00000000 to 0x00FFFFFF is unpredictable.

Virtual Address Space Mapping

Placement of I/O devices in the memory address space (memory-mapped I/O) is an old technique to allow the use any instruction that references memory to manipulate I/O ports at any execution level. The µPD70616 permits a new form of I/O peripheral access which maps a portion of the virtual address space into the I/O address space using the memory management logic.

The I/O mapping process is defined by:

- **unit of mapping**

  The mapping of the virtual address space to the I/O address space is done using the page as the basic unit of mapping. A 4KB segment of the virtual space is I/O mapped by setting the I bit within the corresponding page table entry. The µPD70616 has 4096 I/O pages within the 16MB I/O address space.
- **access type**

  Access to an I/O mapped page is controlled by the read and write permissions defined for the page. An execute access is undefined for an I/O mapped page and if attempted will cause an exception.
- **protection**

  The area table entry execution level permissions operate identically when compared to a memory-mapped page. Both read and write execution levels are independently specified.
- **privilege**

  By I/O mapping a page of virtual space, the area table entries control which tasks can access to the I/O device. This allows non-privileged tasks to perform I/O accesses on a task by task basis as determined by the operating system.

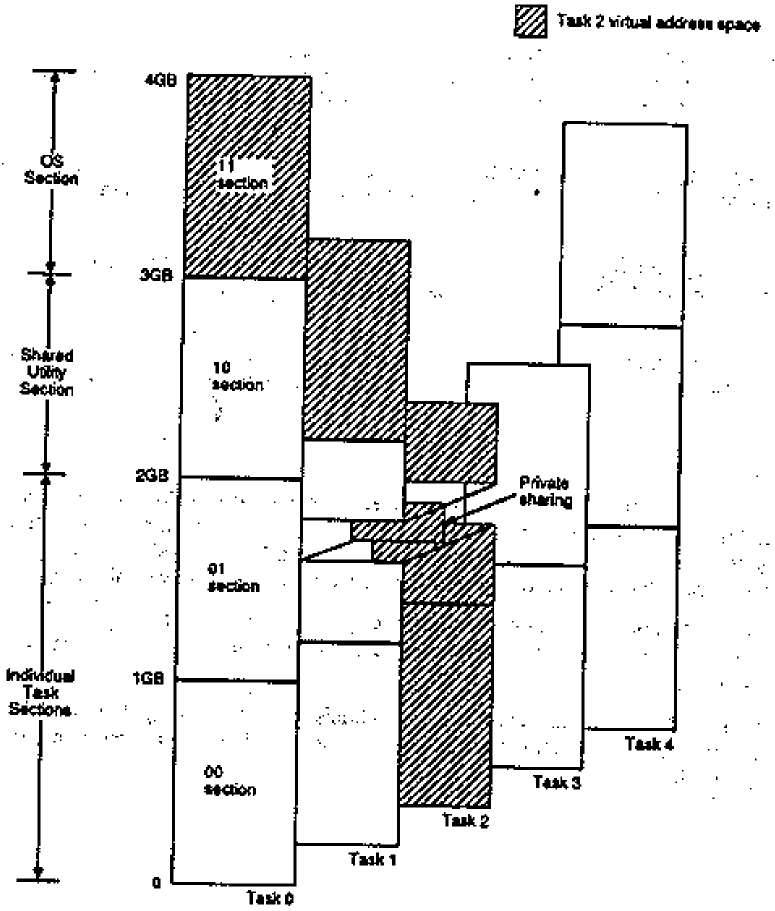
Multiple Virtual Spaces

A multitasking operating system presents a dichotomy. Each installed task must be provided with an independent virtual address spaces for protection and separation from other tasks in the system while at the same time there are common operating system services which are required to be shared by all tasks in the system. The µPD70616 microprocessor solves this dichotomy by allowing tasks to share a portion of their virtual address space.

Virtual address spaces can be shared at the section, area or page level. The operating system virtual space can be shared at the section level by dedicating section 11 (1GB) as a common section and using sections 00, 01 and 10 for a total of 3GB of independent virtual space.

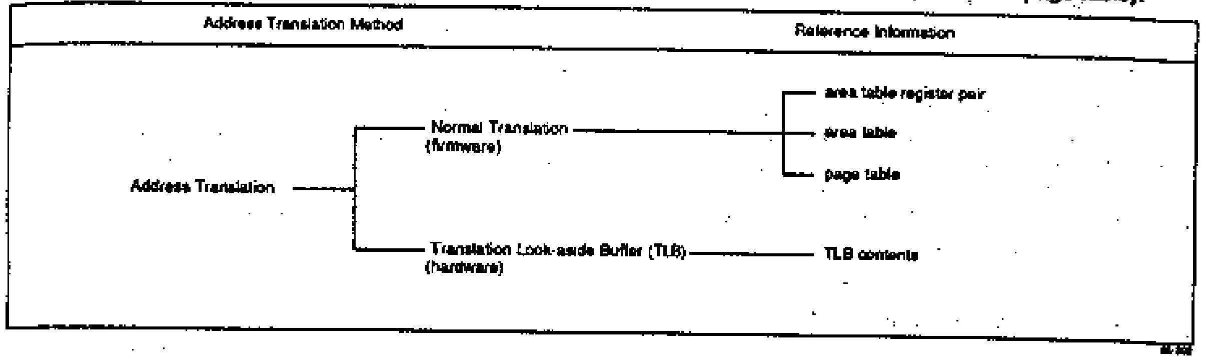
Virtual space need not be shared among all tasks on a global basis. Private sharing of virtual address space (such as a UNIX fork) can be accomplished by using the area as the basis for sharing. An example of a multiple virtual address spaces is shown in figure 4–7.

Figure 4–7. Multiple Virtual Address Spaces



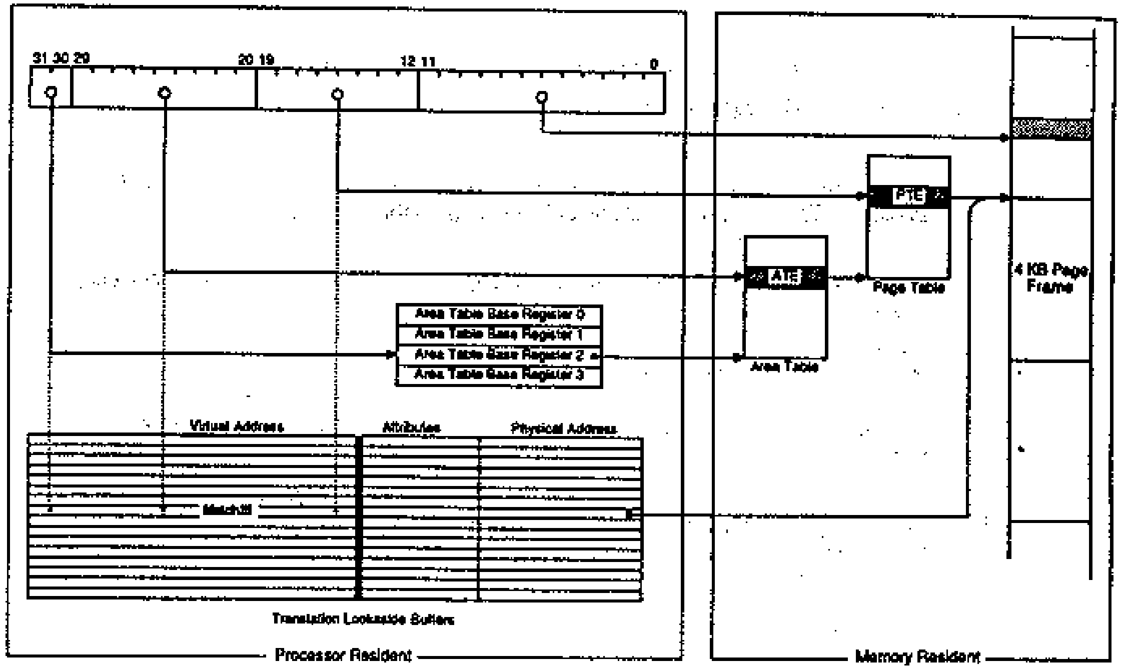
Address Translation

The μPD70616 microprocessor translates virtual addresses using the on-chip memory management unit. Normally, address translation requires the μPD70616 firmware to translate an address using the on-chip memory management registers (area table base and length registers) and the memory resident translation tables (area table/page table).


Address Translation Method
Reference Information
Address Translation
Normal Translation (firmware)
Translation Look-aside Buffer (TLB) (hardware)
area table register pair
area table
page table
TLB contents

Address translation incurs a great deal of overhead and is unacceptable to perform for each access so the results of a translation are cached in a translation look-aside buffer (TLB). Caching of a virtual/physical address pair allows the high speed TLB hardware to perform the translation the next time the page is referenced, eliminating the address translation overhead. An example of the address translation process is shown in figure 4–8.

Figure 4–8. Address Translation



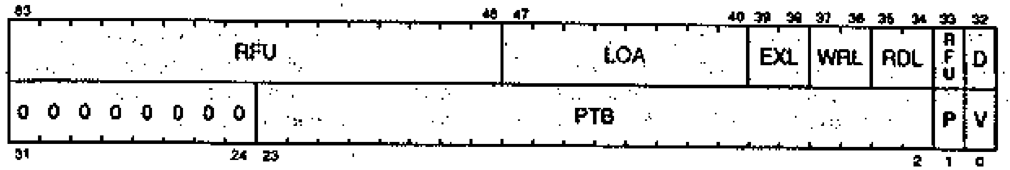
Area Table Register Pair

Associated with each of the four sections is a pair of privileged registers labelled ATBR0/ATLR0 (section 00) to ATBR3/ATLR3 (section 11). Each of area table register pairs contains the base address (physical address) of an associated area table in memory along with length and validity information for the section. The area table registers were described in detail in section 3 of this manual.

Area Tables

Each valid section of a virtual address space has an associated area table. Area tables contain up to 1024 entries which correspond to the area 0 to area 1023. Each entry in an area table is called an area table entry (ATE). An area table entry is eight bytes in size and a complete area table would occupy 8KB of memory. The limit of section (LOS) field in the area table base register permits the section size to be reduced along with size of the corresponding area table. Area tables exist in the memory address space and must aligned on a doubleword boundary. An area table entry is shown figure 4-9.

Figure 4-9. Area Table Entry (ATE)



bit 0 V The V (valid) bit indicates whether the ATE contents are valid.

V = 0 ATE contents are undefined.
V = 1 ATE contents are valid

In the case where the V bit is cleared, the remainder of the ATE is undefined and available. The following definitions only apply when the V bit is set.

bit 1 P The P (present) bit indicates whether the page table specified by the ATE currently exists in memory.

P = 0 page table not present
P = 1 page table present

An Area Not Present Exception will occur if an ATE is referenced during an address translation with the P bit cleared.

bits 2:31 PTB The PTB (page table base) field contains the physical address of a word aligned page table. In the µPD70616 microprocessor, the high order bits of the PTB (bits 24:31) must be zero.

bit 32 D The D (direction) bit specifies the growth direction of the area defined by this ATE.

D = 0 positive area growth direction (increasing addresses)
D = 1 negative area growth direction(decreasing addresses)

bit 33 RFU Reserved for future use

bits 34:35 RDL The RDL (read execution level) field contains the execution level required for read access.

RDL[35:34] =

0	0	level 0
0	1	level 0, 1
1	0	level 0, 1, 2
1	1	level 0, 1, 2, 3

bits 36:37 WRL The WRL (write execution level) field contains the execution level required for write access.

WRL[37:36] =

0	0	level 0
0	1	level 0, 1
1	0	level 0, 1, 2
1	1	level 0, 1, 2, 3

bits 38:39 EXL The EXL (execute execution level) field contains the execution level required for read access.

EXL[39:38] =

0	0	level 0
0	1	level 0, 1
1	0	level 0, 1, 2
1	1	level 0, 1, 2, 3

bits 40:47 LOA The LOA (limit of area) field specifies the range of valid pages for the area defined by this ATE. If the D bit is cleared, pages in the range 0 ≤ N ≤ LOA are defined. If the D is set, pages in the range 255 ≥ N ≥ LOA are defined.

bits 48:63 RFU Reserved for future use

Page Tables

Each valid area of a virtual address space has an associated page table. Page tables contain up to 256 entries which correspond to the page 0 to page 255 within an area. Each entry in a page table is called an page table entry (PTE). Page tables need not be present in memory and can be swapped out to secondary storage if desired by marking the area as not present and allowing the operating system to load the page table and restarting the address translation. When an area is shared by multiple virtual spaces, several ATEs may contain references to the same PTE.

An page table entry is four bytes in size and a complete page table occupies 1KB of memory. The limit of area (LOA) field in the area table entry permits the area size to be reduced along with size of the corresponding page table. Page tables exist in the memory address space and must aligned on a word boundary. An page table entry is shown figure 4-10.

Figure 4-10. Page Table Entry (PTE)

31							24	23 12	11	10	9	8	7	6		3	2	1	0
0	0	0	0	0	0	0	0	RPV	E	W	R	M	A	U	RFU	L	P	I	V

bit 0 V

The V (valid) bit indicates whether the PTE contents are valid.

V = 0	PTE contents are undefined
V = 1	PTE contents are valid

In the case where the V bit is cleared, the remainder of the PTE is undefined and available. The following definitions only apply when the V bit is set. If an address translation using an invalid PTE is attempted, an Invalid Page exception will occur..

bit 1 I

The I (I/O mapped) bit determines if the page specified by this PTE is mapped into the memory or I/O address space.

I = 0	page is not I/O mapped
I = 1	page is I/O mapped

bit 2 P

The P (present) bit indicates whether or not the page specified by this PTE is in memory. A Page Not Present Exception will occur if an address translation is attempted using a PTE with the P bit cleared.

P = 0	page not present
P = 1	page present

If the I bit is set in this PTE, the P bit is undefined and disregarded.

bit 3 L

The L (lock) bit is used to specify when the page is involved in an I/O operation such as a DMA transfer. All CPU accesses to a page marked as locked are prohibited and an Invalid Page Exception will occur if an address translation is attempted using a PTE with the L bit set.

L = 0	page is not locked for I/O
L = 1	page is locked for I/O

This field is undefined if the page is I/O mapped (I = 1).

bits 4:5 RFU

Reserved for future use

bit 6 U

The U (user) field is user definable by the operating system and is ignored during address translation.

bit 7 A

The A (accessed) bit indicates whether the page associated with this PTE has been referenced.

A = 0	not accessed
A = 1	accessed

This field is undefined if the page is I/O mapped (I = 1).

bit 8 M

The M (modified) bit indicates whether a Write access has occurred to the page associated with this PTE.

M = 0	no Write accesses occurred
M = 1	Write access occurred

This field is undefined if the page is I/O mapped (I = 1).

bit 9 R

The R (readable) bit determines if a Read access can be made to the page associated with this PTE.

R = 0	no Read access
R = 1	Read access permitted

bit 10 W

The W (writable) bit determines if a Write access can be made to the page associated with this PTE.

W = 0	no Write access
W = 1	Write access permitted

bit 11 E

The E (executable) bit determines if a Execute access can be made to the page associated with this PTE.

E = 0	no Execute access
E = 1	Execute access permitted

No Execute access is permitted if the page is I/O mapped (I = 1).

bits 12:31 RPN

The RPN (real page number) field has the base address (physical address) of the page associated with this PTE. Pages are aligned on a 4KB page boundary and the lower twelve bits of the physical address are zero. In the µPD70616 microprocessor, the higher order eight bits must be zero. The RPN is a physical address in the memory address space if the I field is cleared, otherwise it is a physical address in the I/O address space.

Section 5
Task Management

A task is the smallest unit of concurrency in a µPD70616 system. In a modern multitasking operating system, multiple tasks can exist simultaneously with the illusion of exclusive control of the processor and system resources. In fact, the operating system is rapidly switching from one task to another based on the relative priorities assigned to each task. At any given moment, one task is the highest priority task and is physically installed on the µPD70616 microprocessor. In a multitasking system there must be a mechanism for the current task to be suspended so that a higher priority task can use the processor.

Context Switching

Each task has a context which completely describes the state of the task. A context switch saves the current context and loads the context of the next task. A task context consists of the following information:

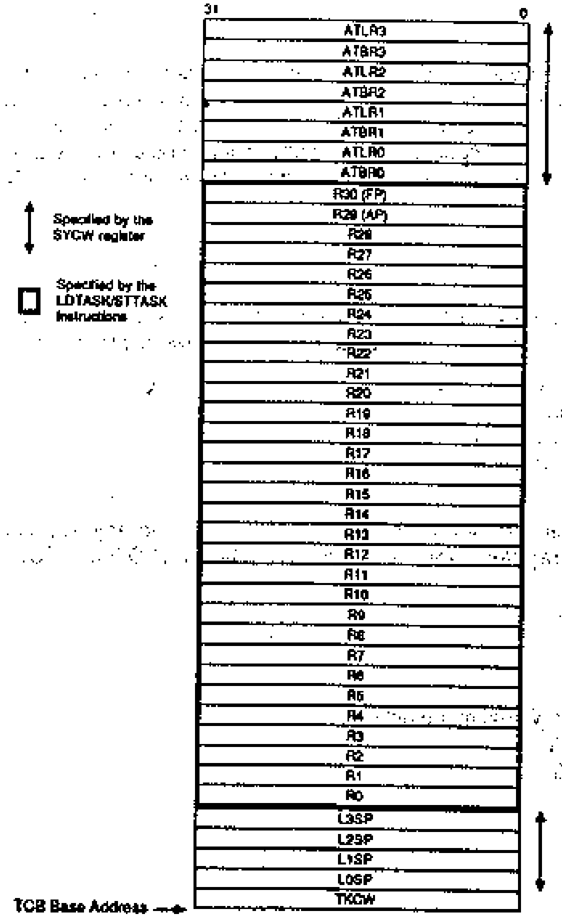
- the program register set
- the virtual address space
- information specific to the task
- other task related information

The task's context is completely defined by a task control block (TCB) and any associated memory management tables. The following is a list of the registers included in the TCB. The full µPD70616 TCB is shown in Figure 5-1.

- Program Register Set
  - R0 – R30
  - L0SP – L3SP
- Memory Management (virtual mode only)
  - ATBR0 – ATBR3
  - ATBL0 – ATBL3
- Task Information
  - TR
  - TKCW

The privileged Task Register (TR) contains the virtual address of the TCB for the current context. The task register is read only and is updated with a new TCB address by the privileged LDTASK instruction.

Figure 5-1. *µPD70616 Task Control Block*

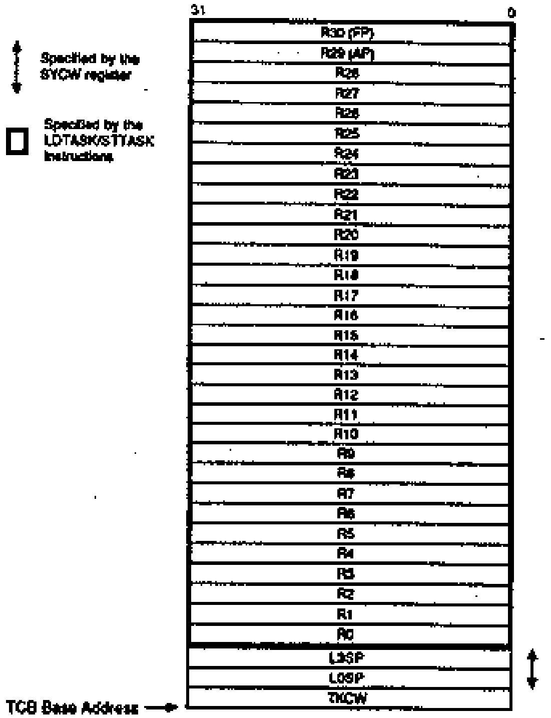


Since not all applications require the use of the full task context, the µPD70616 architecture provides for the elimination of registers from the TCB on a task by task basis. For example, tasks sharing common execution levels and virtual address spaces are not required to maintain the shared registers in their respective task control blocks. Applications requiring fast context switching times can also restrict the size of the general purpose register set. This allows the application designer to trade-off the register set size against the context switch overhead and design a system optimized with respect to some goal.

The inclusion of the area table registers and stack pointers is controlled by information programmed in the System Control Word (SYCW). Control of the size of the general purpose register set is done by specifying a register list operand for the context switch instructions.

An example of a reduced TCB where a single virtual address space and only execution levels 0 and 3 are implemented is shown in Figure 5–2.

Figure 5-2. *Shared Virtual Space TCB*



The real advantage in having these options available is that it makes it possible to customize the µPD70616 context switching to a large range of applications without imposing unnecessary limitations on context switching performance.

Instruction Set Support

The µPD70616 instruction set contains two instructions which support and simplify context switching operations. The Store Task (STTASK) instruction saves the current task context in the TCB defined in the Task Register. The Load Task (LDTASK) instruction sets the Task Register to a new TCB and loads the new context into the processor. These instructions are described in detail in the instruction set section of this manual.

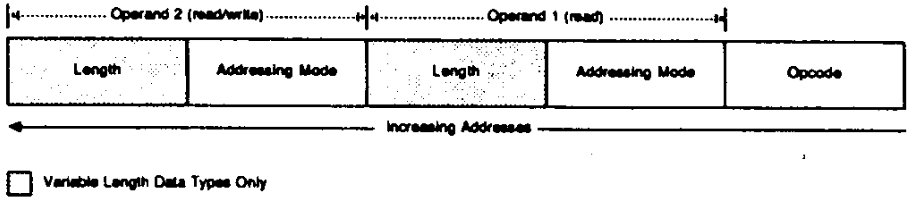
Section 6
Instruction Formats
and Addressing Modes

Instruction Formats

The µPD70616 microprocessor can execute 273 variations of 119 basic instructions. Each of these 119 instructions is encoded using one of seven instruction formats. The instruction format contains useful information on the type of instruction as well as the number and data type of the operands. The µPD70616 instruction formats have been carefully selected and optimized for use in a full 32-bit environment. Considerations such as minimizing the length of frequently executed instructions, a special format for the register/register and register/memory instructions and the use of implicit addressing modes for frequently referenced operands have been employed to reduce the code size and maximize instruction throughput.

It is the purpose of this chapter to describe the structure and format of the binary strings which are referred to as µPD70616 machine code. Instructions and operand addressing information are encoded as binary strings (always an integer number of bytes in length) for the µPD70616 microprocessor to execute. In order to efficiently encode instructions with differing numbers of operands, several different machine instruction formats are provided. The actual format used is determined by the instruction and the operand addressing modes.

In most cases, there is the freedom to use any addressing mode within a given instruction format. However, for reasons of efficiency, one of the two operand formats is required to use a register operand for one of its two operands. Using a two operand instruction as a typical example, a µPD70616 instruction sequence consists of an opcode field followed by addressing information for the first (source) operand followed by the addressing information for the second (destination) operand. In the case of variable length data types, a length field is also provided for the operands.



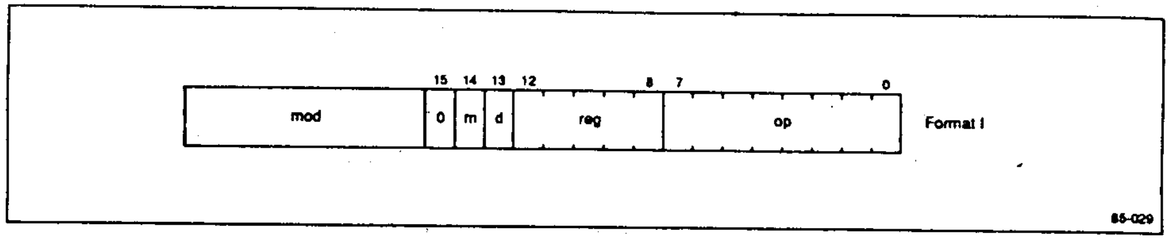
85-001

In many current microprocessor architectures, restrictions are applied to the use of registers, restricting some registers to hold operands and others to hold pointers or indexes. In the µPD70616 architecture, orthogonality is guaranteed by permitting any one of the thirty-two 32-bit general purpose registers to contain a data operand, a pointer (base register) or an index (index register) value. The orthogonal µPD70616 instruction set means that with but a few exceptions, there is complete freedom to use any addressing mode with any operand. Source operands may be referenced using any one of the twenty-one byte addressing modes (eighteen bit addressing modes) while destination operands, for obvious reasons, are restricted from using the immediate addressing modes.

The remainder of this section is devoted to describing the instruction formats and addressing modes used by the µPD70616 microprocessor. Condensed versions of this information is also summarized in Appendix B and Appendix C.

Instruction Format I

Instruction Format I is an optimized format used for two operand instructions in the register/register and register/memory operand classes. The minimum instruction length of this instruction format is three bytes, but it may be extended to greater length, depending upon the addressing mode field of the second operand.



bits 0:7 op The opcode field specifies the instruction to be executed.

bits 8:12 reg This field identifies the register to be used as the source or destination operand for this instruction.

bit 13 d The direction bit specifies whether the register field is treated as the source or destination operand as follows:

d = 0 reg field identifies the source operand
d = 1 reg field identifies the destination operand

bit 14 m This field is used in determining the addressing mode of the second operand.

bit 15 0 This field is cleared in all Format I instructions.

mod This field specifies the addressing mode for the second operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

The hexidecimal representation used for the opcode field of Format I instructions is

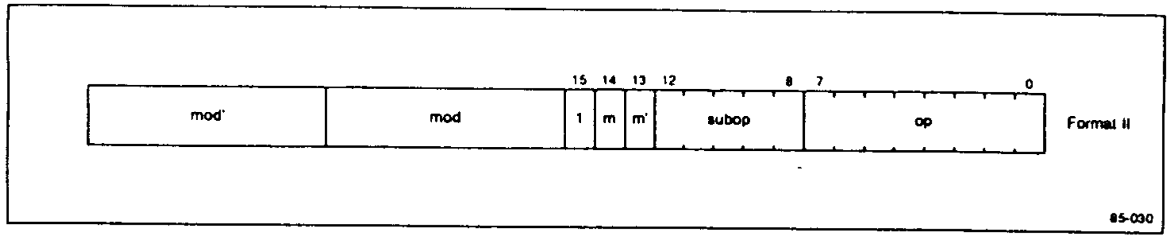
op

where op is the eight bit wide opcode field. An example of this format is the integer addition instruction which is represented as

Instruction	Opcode
add.b	80

Instruction Format II

Instruction Format II is a more general form of Format I and can be used for two operand instructions using any addressing modes or when the opcode field must be extended as in the floating point arithmetic instructions. The minimum instruction length of a Format II instruction is four bytes, but may be extended to greater length, depending upon the addressing modes of the operands.



bits 0:7 op The opcode field specifies the instruction to be executed.

bits 8:12 subop This field specifies additional opcode information for extended opcode instructions.

bit 13 m' This field is used in determining the addressing mode of the second operand.

bit 14 m This field is used in determining the addressing mode of the first operand.

bit 15 1 This field is set in all Format II instructions.

mod This field specifies the addressing mode for the first operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

mod' This field specifies the addressing mode for the second operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

The hexidecimal representation used for the opcode field of Format II instructions is

op	Format I/II instructions
op•subop	Format II instructions

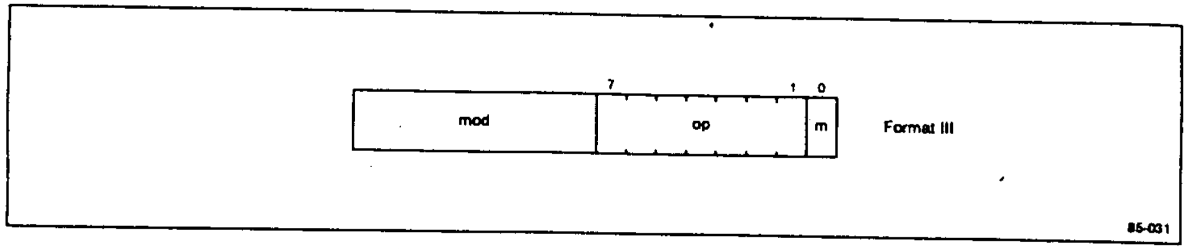
where op is the eight bit wide opcode field and subop is the five bit wide opcode extension field. A typical example of this format is the floating point addition instruction.

Instruction	Opcode
addf.s	5C•18

Unless otherwise specified, the subop field of a Format I/II instruction is zero.

Instruction Format III

Instruction Format III is used for instructions accepting a single operand. The minimum length of this instruction format is two bytes, but it may be extended to greater length, depending upon the selected addressing mode of the operand.



bit 0	m	This field is used in determining the addressing mode of the operand.
bits 1:7	op	The opcode field specifies the instruction to be executed.
	mod	This field specifies the addressing mode for the specified operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

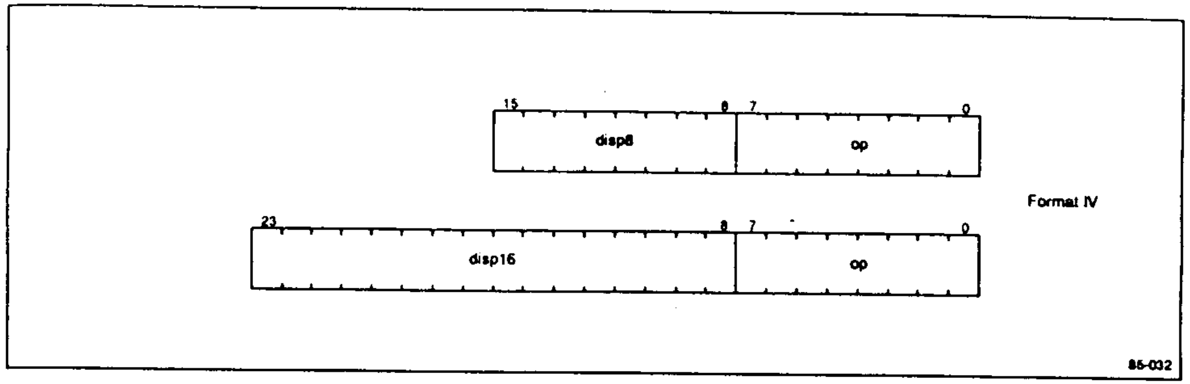
An example of the hexidecimal representation used for the opcode field of Format III instructions is

<u>Instruction</u>	<u>Opcode</u>
pop	E6/7

where the value of the second digit of the opcode field is determined by the addressing mode.

Instruction Format IV

Instruction Format IV is used for instructions which use the PC relative addressing mode. The instruction length of this format is two or three bytes, depending on the size of the displacement field.



bits 0:7	op	The opcode field specifies the instruction to be executed.
bits 8:15(23)	disp	This field contains the 8/16-bit displacement which is sign extended to 32 bit length and added to the contents of the PC to compute the next PC.

The hexidecimal representation used for the opcode field of Format IV instructions is

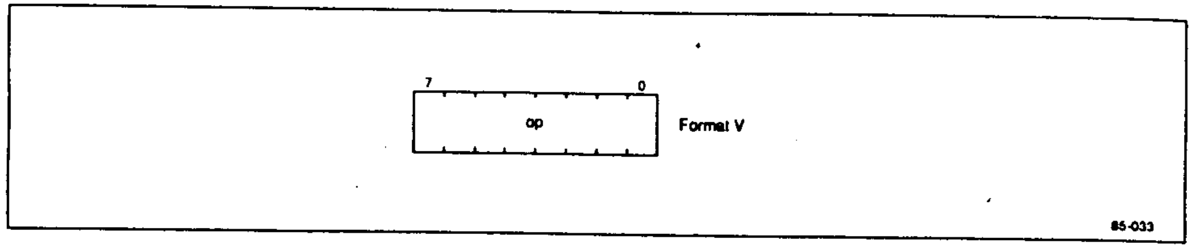
op

where op is the eight bit wide opcode field. An example of this format is the branch to subroutine instruction which is represented as

<u>Instruction</u>	<u>Opcode</u>
bsr	48

Instruction Format V

Instruction Format V is used for instructions which take no operands. The instruction length of these instructions is one byte.



bits 0:7 op The opcode field specifies the instruction to be executed.

The hexidecimal representation used for the opcode field of Format V instructions is

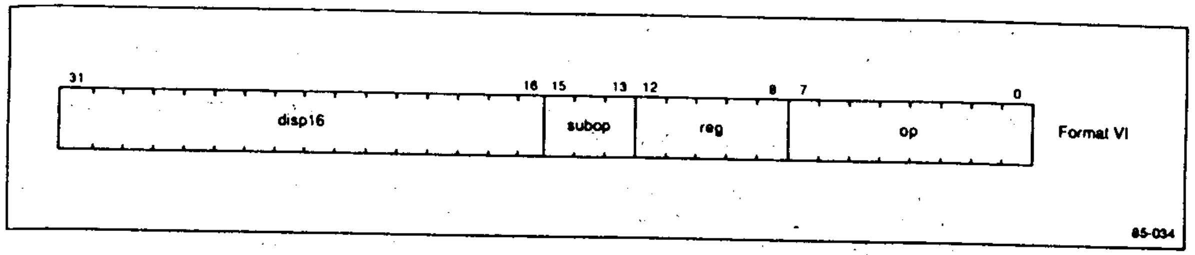
op

where op is the eight bit wide opcode field. An example of this format is the NOP instruction which is represented as

Instruction	Opcode
nop	CD

Instruction Format VI

Instruction Format VI is used for the loop instructions and contains both control register and displacement fields. The length of instruction format VI instructions is four bytes.



bits 0:7 op The opcode field specifies the instruction to be executed.

bits 8:12 reg This field identifies the register to be used as the control register.

bit 13:15 subop This field contains additional instruction identification information.

bits 16:31 disp16 This field contains the 16-bit displacement which is sign extended to 32 bit length and used as an offset from the current contents of the PC.

The hexidecimal representation used for the opcode field of Format VI instructions is

op•subop.

where op is the eight bit wide opcode field and subop is a three bit wide opcode extension field. An example of this format is the decrement and branch if not zero instruction.

Instruction	Opcode
dbnz	C7•2

Instruction Format VII

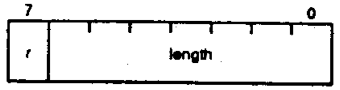
Instruction Format VII is used for instructions manipulating variable length data types such as character strings and bit strings as well as the decimal arithmetic instructions. Format VII instructions are subdivided into three subtypes depending the number and location of the variable length operands.

Format VIIa Used with instructions that reference two variable length operands.

Format VIIb Used when the source operand is a variable length data type and the destination operand is a fixed length data type.

Format VIIc Used when the source operand is a fixed length data type and the destination operand is a variable length data type or with decimal arithmetic instructions.

Format VII instructions contain an 8-bit extension field which is used to determine the length of a variable length character or bit string operand. The most significant bit of the extension field is used to determine whether the direct mode (the operand length is in the lower seven bits of the extension field) or the indirect mode (the operand length is contained in the general purpose register identified by the lower seven bits of the extension field) is specified. This field is also used to store the mask pattern for the ADDDC, SUBDC, SUBRDC, and CVTD instructions.



86-096

bit 7 r The r (register) bit determines whether the length field contains the operand length (direct mode) or contains a pointer to a general purpose register containing the operand length. This field is decoded as follows:

r = 0 Direct mode, length field contains the operand length.

r = 1 Indirect mode, length field contains the number of a general purpose register (0–31) that contains the operand length.

bits 6:0 length The length operand (0–127) or the register ID (0–31) resides in this field, as determined by the r field.

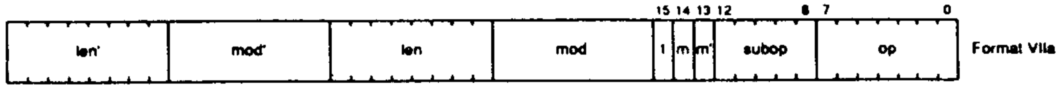
All Format VII instructions use a 12-bit instruction field. The hexidecimal representation used for the opcode field of Format VII instructions is

op•subop

where op is the eight bit wide opcode field and subop is the five bit wide opcode extension field. A typical example of this format is the AND bit string (upward) instruction.

Instruction	Opcode
andbsu	5B•10

Format VIIa



85-035

bits 0:7 op The opcode field specifies the instruction to be executed.

bits 8:12 subop This field specifies additional opcode information for the instruction.

bit 13 m' This field is used in determining the addressing mode of the second operand.

bit 14 m This field is used in determining the addressing mode of the first operand.

bit 15 1 This field is set in all Format VIIa instructions.

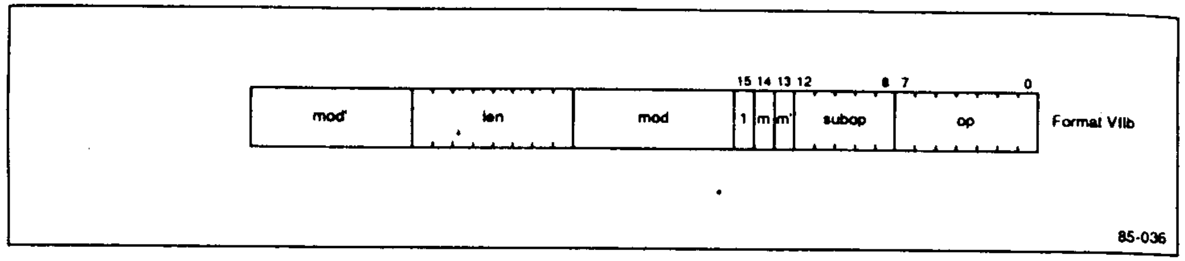
mod This field specifies the addressing mode for the first operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

len First operand length field.

mod' This field specifies the addressing mode for the second operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

len' Second operand length field.

Format VIIb



bits 0:7 op The opcode field specifies the instruction to be executed.

bits 8:12 subop This field specifies additional opcode information for the instruction.

bit 13 m' This field is used to determine the addressing mode of the second operand.

bit 14 m This field is used to determine the addressing mode of the first operand.

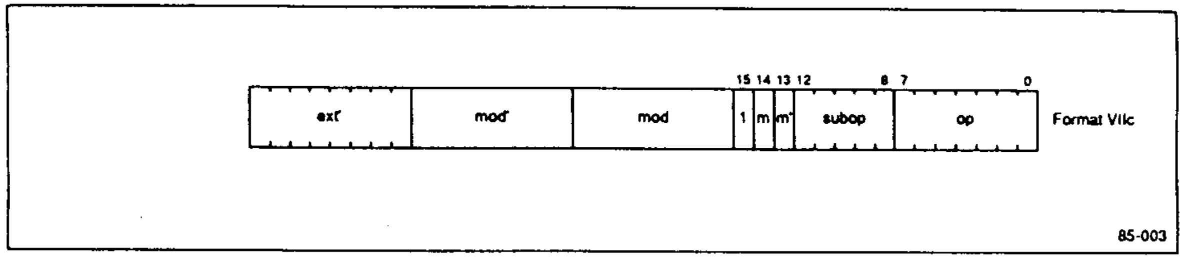
bit 15 1 This field is set in all Format VIIb instructions.

mod This field specifies the addressing mode for the first operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

len Variable length operand length field.

mod' This field specifies the byte addressing mode for the second operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

Format VIIc



bits 0:7 op The opcode field specifies the instruction to be executed.

bits 8:12 subop This field specifies additional opcode information for the instruction.

bit 13 m' This field is used in determining the addressing mode of the second operand.

bit 14 m This field is used in determining the addressing mode of the first operand.

bit 15 1 This field is set in all Format VIIc instructions.

mod This field specifies the addressing mode for the first operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

mod' This field specifies the addressing mode for the second operand. This field ranges in length from 1 to 9 bytes depending on the specified addressing mode.

ext' Operand extension field containing either a length operand or a mask pattern (decimal arithmetic instructions).

Addressing Modes

The µPD70616 microprocessor provides fourteen basic addressing modes for referencing byte aligned data. These basic addressing modes include:

- Register
- Autoincrement
- Displacement
- Displacement Indirect
- Double Displacement
- Direct Address
- Immediate
- Register Indirect
- Autodecrement
- PC Displacement
- PC Displacement Indirect
- PC Double Displacement
- Direct Address Deferred
- Immediate Quick

Registers are often used to hold an index to select a particular member of an array. Unfortunately, an index is the same as an offset only when the members of an array occupy exactly one byte. The µPD70616 architecture offers a number of scaled index addressing modes to help ease this problem with the management of indexes. With seven of the basic addressing modes, it is possible to specify an additional index register. The contents of an index register is multiplied by the size of the operand (in bytes) and added to the base address. Thus, any of these seven addressing modes can be used to specify the base address of an array, and then use the index register (which can be any of the 32 general purpose registers) to index into the array.

The value used to scale a register in the autoincrement, autodecrement and scaled index addressing modes depends on the data type of the operand. The following table summarizes this information:

Data Type	Scaling Constant	
	Increment/Decrement	Scaled Index
Byte	1	1
Halfword	2	2
Word	4	3
Doubleword	8	8
Short Real	4	4
Long Real	8	8
Packed Decimal	1	1
Unpacked Decimal	2	2
Byte Character	1	1
Halfword Character	2	2
Bit	4	4
Bit Field	4	–
Bit String	1	–

– Not Available

86-020

The µPD70616 also supports eighteen bit addressing modes for addressing bit-aligned data types such as bit fields and bit strings. Bit addressing modes enable the programmer to specify a byte base address and a 32-bit bit offset in a single operand.

The notations for these twenty-one byte addressing modes and the eighteen bit addressing modes are summarized in the following table.

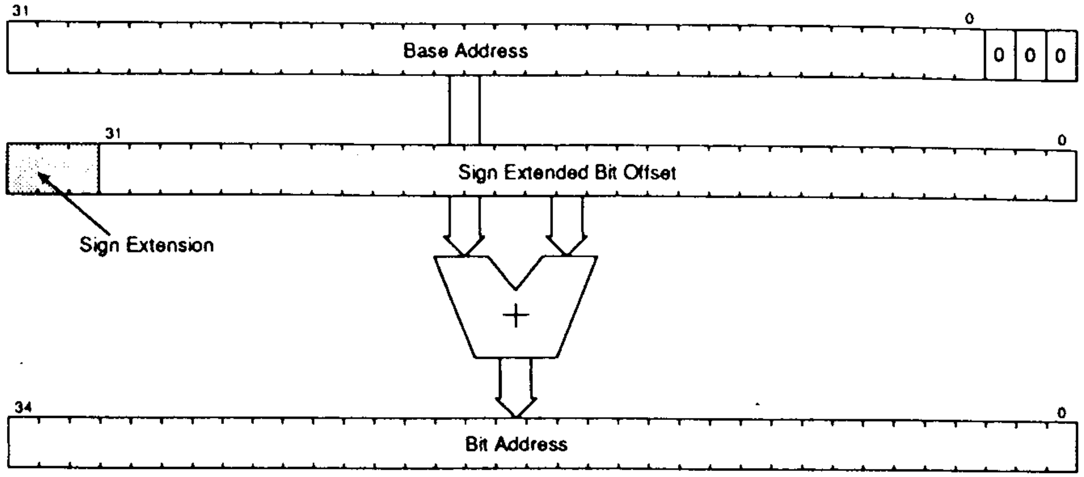
Addressing Mode	Syntax	
	Byte Addressing	Bit Addressing
Register	Rn	–
Register Indirect	[Rn]	@[Rn]
Register Indirect Indexed	[Rn](Rx)	Rx@[Rn]
Autoincrement	[Rn+]	@[Rn+]
Autodecrement	[–Rn]	@[–Rn]
Displacement	disp [Rn]	offset@[Rn]
PC Displacement	disp [PC]	offset@[PC]
Displacement Indexed	disp [Rn](Rx)	Rx@disp[Rn]
PC Displacement Indexed	disp [PC](Rx)	Rx@disp[PC]
Displacement Indirect	[disp [Rn]]	@[disp [Rn]]
PC Displacement Indirect	[disp [PC]]	@[disp [PC]]
Displacement Indirect Indexed	[disp [Rn]](Rx)	Rx@[disp [Rn]]
PC Displacement Indirect Indexed	[disp [PC]](Rx)	Rx@[disp [PC]]
Double Displacement	disp1 [disp2 [Rn]]	offset@[disp [Rn]]
PC Double Displacement	disp1 [disp2 [PC]]	offset@[disp [PC]]
Direct Address	/addr	@/addr
Direct Address Indexed	/addr (Rx)	Rx@/addr
Direct Address Deferred	[/addr]	@[/addr]
Direct Address Deferred Indexed	[/addr](Rx)	Rx@[/addr]
Immediate	#value	–
Immediate Quick	#value (1–15)	–

– Not Available

86-097

Calculation of Bit Addresses

A virtual address for byte aligned data consists of 32 bits. Since the identification of a particular bit within a byte requires an additional three bits, a total of 35 bits is needed to address an arbitrary bit in virtual memory. This means that the address of a bit cannot be expressed in a single 32-bit word. Two 32-bit words can be used to form a bit address, and a convenient convention for this is to interpret one of the words as a byte base address and the other as a bit offset from the base address. To form the necessary 35-bit bit address, the byte base address is zero extended on the right to form a 35-bit base address. The 32-bit bit offset is then sign extended to 35 bit length and added to the base address. The 35-bit sum is then used to address the start of the bit data type in memory.



Eighteen of the byte addressing modes (the exceptions are register and the two immediate addressing modes) discussed in the preceding section require two 32-bit values (often one of the values is implicitly 0) to be added in order to obtain the address of the operand. The μPD70616 re-interprets these addressing modes as suggested by the above diagram for instructions which operate on bit fields and bit strings. The bit addressing modes can be classified into bit displacement and bit index modes.

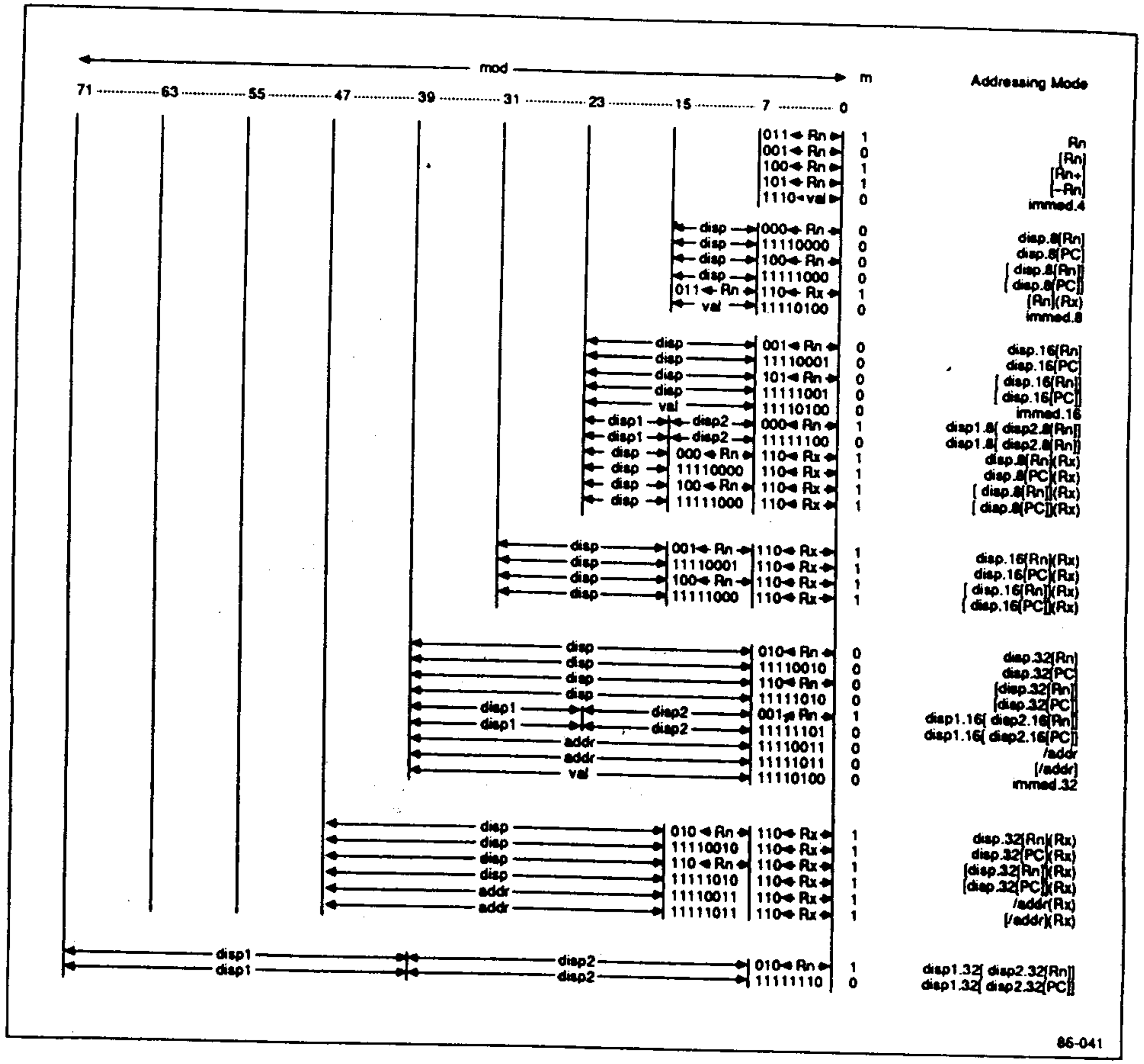
The bit displacement modes are re-interpretations of the byte displacement addressing modes. The displacement field is interpreted as the bit offset from the base address. In the case of instructions with no displacement field, an offset of 0 is substituted. The other exception is the double displacement addressing mode which uses one displacement field to locate a memory based pointer and the second displacement field as the bit offset.

The bit index modes are re-interpretations of the scaled index addressing modes. When used where a bit address is required, the index register Rx is interpreted as a bit displacement and any base register and displacement fields form the byte base address. Notice that this addressing mode is not inconsistent with its other uses, since the index register is scaled to the size of the data, which in this case is a single bit.

Addressing Mode Encoding

The μPD70616 addressing modes for denoting byte aligned data are described in the next section. The minimum encoding of any of the fourteen basic addressing modes requires nine bits. It is convenient to divide these nine bits into three fields, mod, Rn and m.

The m and mod fields together define the addressing mode. The five bit Rn field is used to specify a register number, or it may provide other information if the addressing mode uses no register as in the case of the PC relative and immediate addressing modes. The scaled index extension modes all require an index register, Rx, in addition to the base register, Rn. Five bits are needed to specify the register itself, and three extra bits beyond that are used to indicate the fact that the scaled index addressing mode is being used. The placement of these additional eight bits is shown in the diagram below.


mod
m
Addressing Mode
71 63 55 47 39 31 23 15 7 0
011 Rn 1 Rn
001 Rn 0 [Rn]
100 Rn 1 [Rn+]
101 Rn 1 [-Rn]
1110 val 0 immed.4
disp 000 Rn 0 disp.8[Rn]
disp 11110000 0 disp.8[PC]
disp 100 Rn 0 [disp.8[Rn]]
disp 11111000 0 [disp.8[PC]]
011 Rn 110 Rx 1 [Rn](Rx)
val 11110100 0 immed.8
disp 001 Rn 0 disp.16[Rn]
disp 11110001 0 disp.16[PC]
disp 101 Rn 0 [disp.16[Rn]]
disp 11111001 0 [disp.16[PC]]
val 11110100 0 immed.16
disp1 disp2 000 Rn 1 disp1.8[disp2.8[Rn]]
disp1 disp2 11111100 0 disp1.8[disp2.8[Rn]]
disp 000 Rn 110 Rx 1 disp.8[Rn](Rx)
disp 11110000 110 Rx 1 disp.8[PC](Rx)
disp 100 Rn 110 Rx 1 [disp.8[Rn]](Rx)
disp 11111000 110 Rx 1 [disp.8[PC]](Rx)
disp 001 Rn 110 Rx 1 disp.16[Rn](Rx)
disp 11110001 110 Rx 1 disp.16[PC](Rx)
disp 100 Rn 110 Rx 1 [disp.16[Rn]](Rx)
disp 11111000 110 Rx 1 [disp.16[PC]](Rx)
disp 010 Rn 0 disp.32[Rn]
disp 11110010 0 disp.32[PC]
disp 110 Rn 0 [disp.32[Rn]]
disp 11111010 0 [disp.32[PC]]
disp1 disp2 001 Rn 1 disp1.16[disp2.16[Rn]]
disp1 disp2 11111101 0 disp1.16[disp2.16[PC]]
addr 11110011 0 /addr
addr 11111011 0 [/addr]
val 11110100 0 immed.32
disp 010 Rn 110 Rx 1 disp.32[Rn](Rx)
disp 11110010 110 Rx 1 disp.32[PC](Rx)
disp 110 Rn 110 Rx 1 [disp.32[Rn]](Rx)
disp 11111010 110 Rx 1 [disp.32[PC]](Rx)
addr 11110011 110 Rx 1 /addr(Rx)
addr 11111011 110 Rx 1 [/addr](Rx)
disp1 disp2 010 Rn 1 disp1.32[disp2.32[Rn]]
disp1 disp2 11111110 0 disp1.32[disp2.32[PC]]
86-041

The remainder of this section describes in detail the μPD70616 addressing modes. A sample of the format used to describe each of the addressing modes is shown below.

Addressing Mode Format

Format This field describes the assembly language format for the byte (bit) addressing modes.

Description

A description of the byte (bit) addressing mode appears in this area.

mod Field Encoding

The format of the mod field is contained in this section. Individual fields can be identified using the following keys:

Rn	Register n
Rx	Index register x
disp	Displacement
val	Immediate value
addr	32-bit address

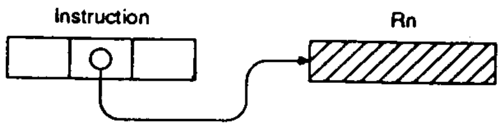
Notes Any additional information relevant to the addressing mode is supplied in this section.

Register Rn

Format Rn.............Byte Addressing

Description

Byte The operand is found in the specified register (or pair of consecutive registers). This addressing mode allows the contents of any of the 32 general purpose registers to be used as the operand in an instruction. For byte or halfword data, the low order portion of the register is used (bits 7:0 or 15:0). For doubleword (64-bit) data, the operand resides in the registers Rn and Rn+1, with the least significant word located in register Rn. The use of R31 for doubleword data is unpredictable.



Bit Prohibited

mod Field Encoding

```
     ←—— mod ——→     m
     7          0
Rn   0 1 1 ←—Rn—→    1
```

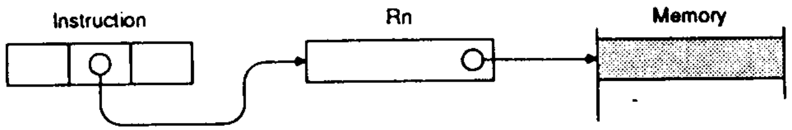
Notes The use of this addressing mode with a variable length data type will generate an Illegal Addressing Mode exception.

Register Indirect [Rn]

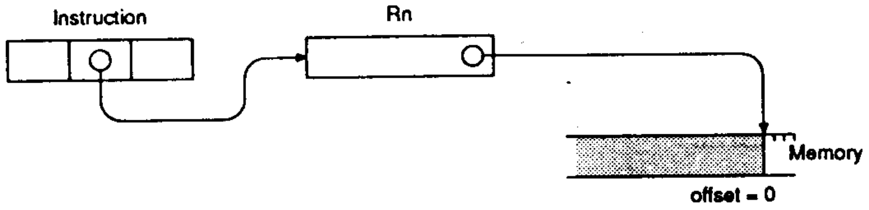
Format [Rn]...........Byte Addressing @[Rn].............Bit Addressing

Description

Byte This addressing mode allows the contents of any general purpose register to be used as a pointer to an operand located in memory.



Bit The base address of the operand in Rn is combined with a default bit offset of 0 to generate the bit address of the operand.



mod Field Encoding

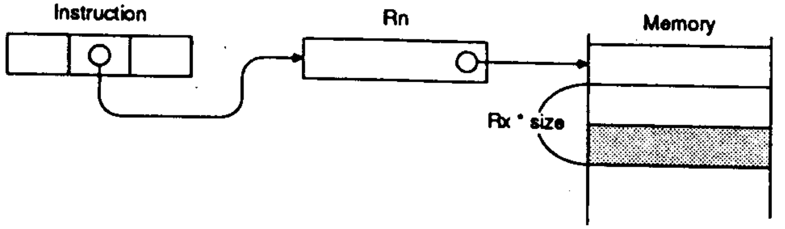
```
       ←—— mod ——→     m
       7          0
[Rn]   0 1 1 ←—Rn—→    0
```

Register Indirect Indexed [Rn](Rx)

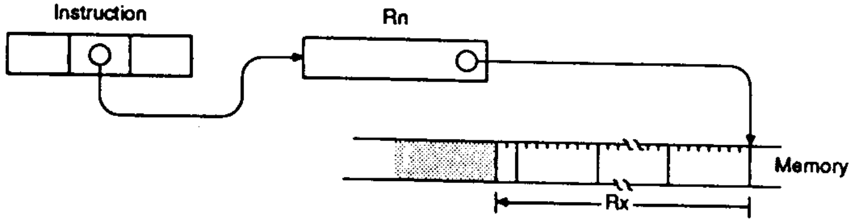
Format [Rn](Rx)......Byte Addressing Rx@[Rn]..........Bit Addressing

Description

Byte This addressing mode is similar to the register indirect mode. The contents of a general purpose register is used as a pointer to a word location which is scaled by the contents of the specified index register.



Bit The index register Rx is is used as the bit offset from the byte base address contained in Rn.



mod Field Encoding

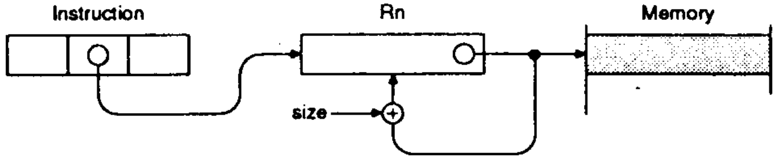
```
         ←——— mod ———→          m
   15         7          0
[Rn](Rx)  0 1 1 ←Rn→ 1 1 0 ←Rx→  1
```

Autoincrement [Rn+]

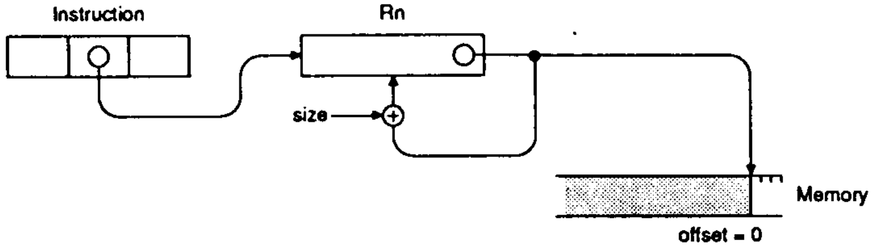
Format [Rn+]..........Byte Addressing @[Rn+]............Bit Addressing

Description

Byte This mode is similar to the register indirect mode, except that the contents of the register is incremented, after the access is made, by the size (in bytes) in the operand. Any of the 32 general purpose registers may be used for this addressing mode.



Bit The contents of register Rn and a bit offset of 0 are used to compute the bit address of the operand. The contents of Rn are then incremented by 1 for the bit string data type or by 4 for the bit field data type.



mod Field Encoding

```
      ←— mod —→    m
      7        0
[Rn+]  1 0 0 ←Rn→  1
```

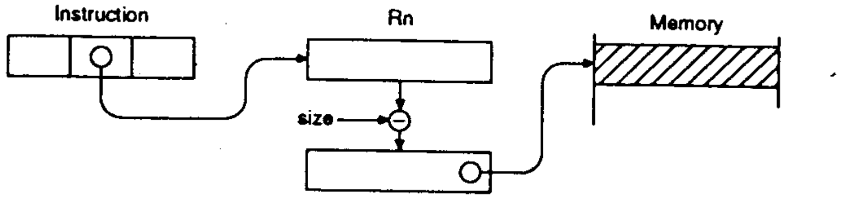
Notes The use of the autoincrement addressing mode in combination with certain other addressing modes may lead to unpredictable results. In particular, if the first operand of an instruction uses the autoincrement mode, the second operand should not use the same register for an index register. Moreover, two operands of an instruction should both not use the same register in the autoincrement (or autodecrement) mode. For further information, see the description of the addressing mode restrictions at the end of this section.

Autodecrement [–Rn]

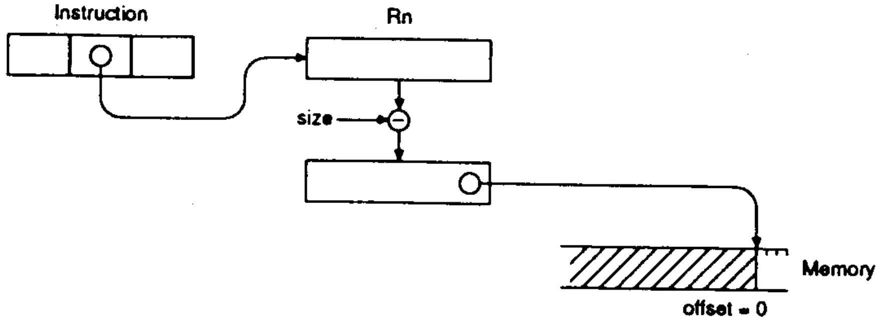
Format [–Rn].........Byte Addressing @[–Rn]............Bit Addressing

Description

Byte This addressing mode is similar to the register indirect mode, except that the contents of the register is decremented, before the operand access is made, by the size (in bytes) of the operand. Any of the 32 general purpose registers may be used with this addressing mode.



Bit The contents of base register Rn is first decremented (count determined by the data type) and the result is added to a bit offset of 0 to generate the operand bit address.



mod Field Encoding

	7 ← mod → 0	m
[–Rn]	1 0 1 ←Rn→	1

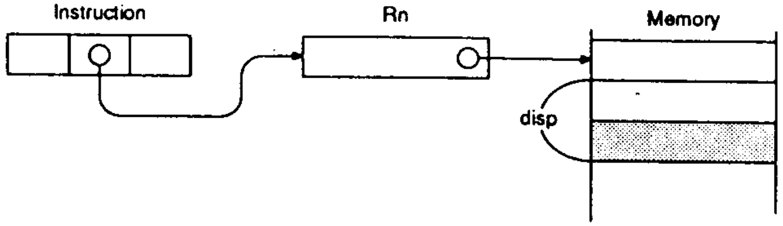
Notes The use of the autodecrement addressing mode in combination with certain other addressing modes may lead to unpredictable results. In particular, if the first operand of an instruction uses the autodecrement mode, the second operand should not use the same register for an index register. Moreover, two operands of an instruction should both not use the same register in the autodecrement (or autoincrement) mode. For further information, see the description of the addressing mode restrictions at the end of this section.

Displacement disp[Rn]

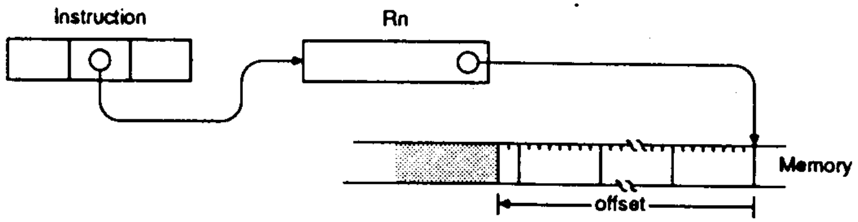
Format disp[Rn].....Byte Addressing offset@[Rn].....Bit Addressing

Description

Byte The 8-, 16-, or 32-bit displacement is sign extended (if 8- or 16-bits) to 32-bit length and added to the contents of the specified register. The resulting value is then used as the address of the operand.



Bit The 8-, 16-, or 32-bit bit offset is combined with the contents of base register Rn to generate the bit address.



mod Field Encoding

← mod → m

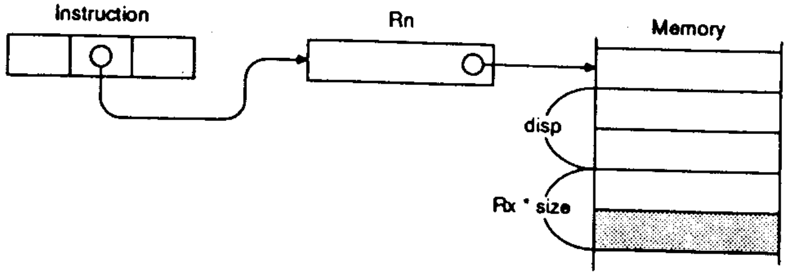
	39	23	15	7	0	m
disp8[Rn]			←disp8→	0 0 0	←Rn→	0
disp16[Rn]		←disp16→		0 0 1	←Rn→	0
disp32[Rn]	←disp32→			0 1 0	←Rn→	0

Displacement Indexed disp[Rn](Rx)

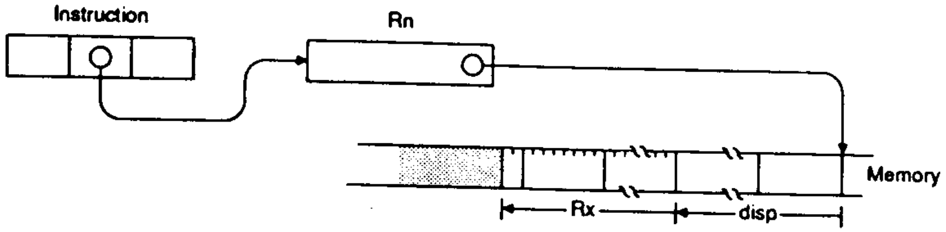
Format disp[Rn](Rx).....Byte Addressing Rx@disp[Rn]....Bit Addressing

Description

Byte The operand located at the sum of the base register Rn, the displacement field and the scaled contents of index register Rx is addressed.



Bit The base address of the operand is formed by the sum of the base register Rn and the displacement field. This value is combined with the bit offset in register Rx to form the operand bit address.



mod Field Encoding

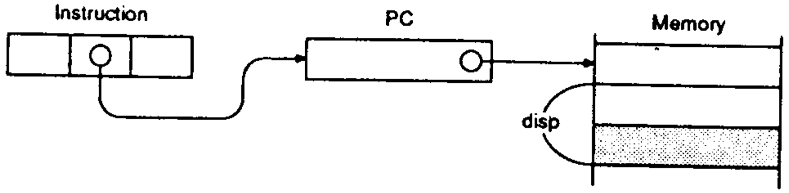
```
              ←──────────────────────── mod ────────────────────────→   m
              47          31          23          15          7          0
disp8[Rn](Rx)                         ←─disp8─→   0 0 0  ←─Rn─→  1 1 0  ←─Rx─→   1
disp16[Rn](Rx)            ←──────disp16───────→   0 0 1  ←─Rn─→  1 1 0  ←─Rx─→   1
disp32[Rn](Rx) ←──────────disp32──────────────→   0 1 0  ←─Rn─→  1 1 0  ←─Rx─→   1
```

PC Displacement disp[PC]

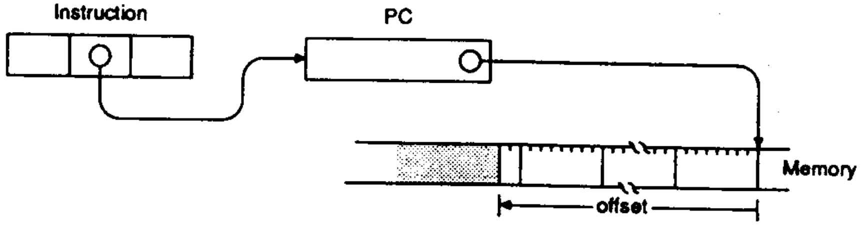
Format disp[PC].....Byte Addressing offset@[PC].....Bit Addressing

Description

Byte This addressing mode permits the program counter to be used in the place of one of the 32 general purpose registers in the displacement mode. The use of the PC displacement addressing mode generates position independent code, since the displacement from the PC is the same without regard to the actual value of the PC.



Bit The contents of the PC is used as the base address and is combined with the offset field to generate the bit address of the operand.



mod Field Encoding

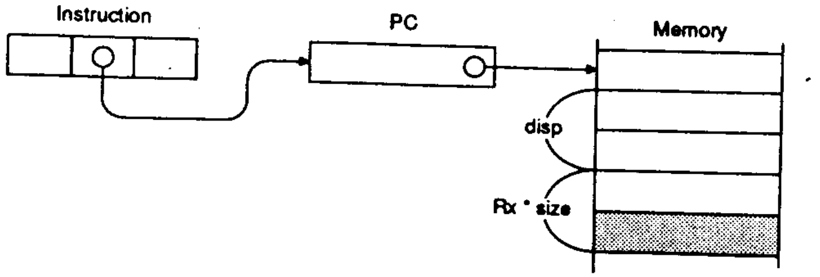
```
             ←──────────────── mod ────────────────→   m
             39          23          15          7              0
disp8[PC]                            ←─disp8─→1 1 1 1 0 0 0 0   0
disp16[PC]               ←────disp16─────────→1 1 1 1 0 0 0 1   0
disp32[PC]   ←──────────disp32───────────────→1 1 1 1 0 0 1 0   0
```

PC Displacement Indexed disp[PC](Rx)

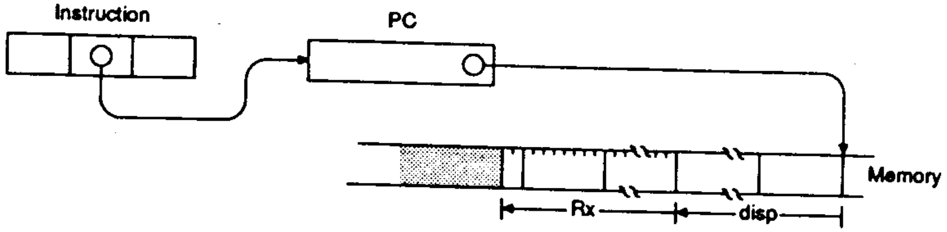
Format disp[PC](Rx).....Byte Addressing Rx@disp[PC]....Bit Addressing

Description

Byte The sum of the PC, the displacment field and the scaled contents of index register Rx is used as the address of the operand.



Bit The sum of the PC and displacement field is used as the base address and is combined with the bit offset in register Rx to form the operand bit address.



mod Field Encoding

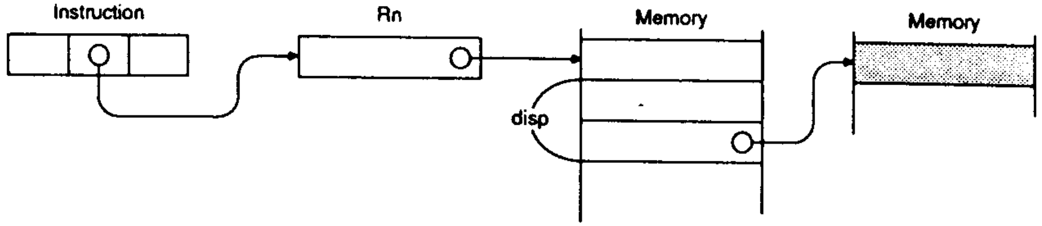
	← mod → (47 31 23 15 7 0)	m
disp8[PC](Rx)	←disp8→ 1 1 1 1 0 0 0 0 110 ←Rx→	1
disp16[PC](Rx)	←disp16→ 1 1 1 1 0 0 0 1 110 ←Rx→	1
disp32[PC](Rx)	←disp32→ 1 1 1 1 0 0 1 0 110 ←Rx→	1

Displacement Indirect [disp[Rn]]

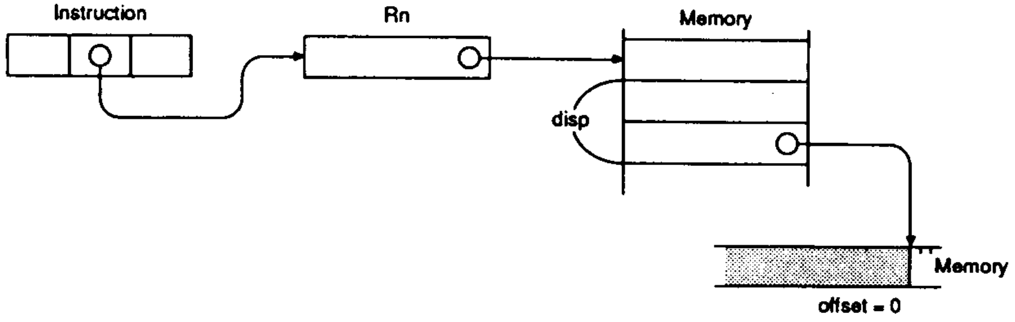
Format [disp[Rn]].........Byte Addressing @[disp[Rn]].......Bit Addressing

Description

Byte The word contents of the location addressed by the sum of the base register Rn and the 8-, 16-, or 32-bit sign extended displacement field is used as the address of the operand.



Bit The word contents of the location addressed by the sum of the base register Rn and the 8-, 16-, or 32-bit sign extended displacement field is used as the bit address of the bit operand. No bit offset is specified in this addressing mode.



mod Field Encoding

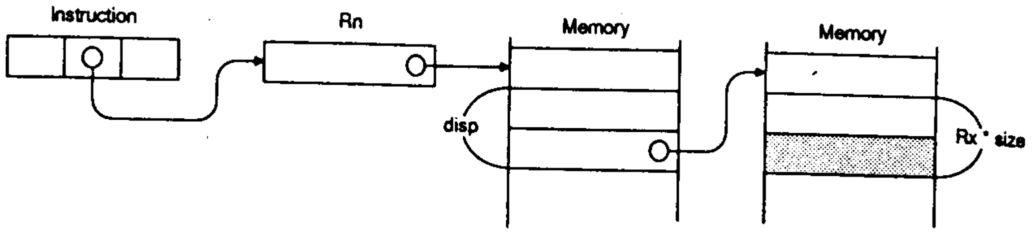
	← mod → (39 23 15 7 0)	m
[disp8[Rn]]	←disp8→ 1 0 0 ←Rn→	0
[disp16[Rn]]	←disp16→ 1 0 1 ←Rn→	0
[disp32[Rn]]	←disp32→ 1 1 0 ←Rn→	0

Displacement Indirect Indexed [disp[Rn]](Rx)

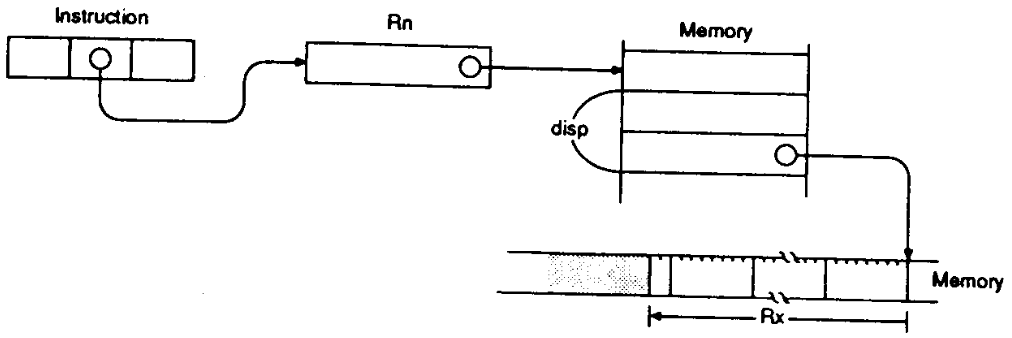
Format [disp[Rn]](Rx).........Byte Addressing Rx@[disp[Rn]].............Bit Addressing

Description

Byte The word contents of the memory location addressed by the sum of the base register Rn and the displacement field is added to the scaled contents of index register Rx and used as the address of the operand.



Bit The word contents of the memory location addressed by the sum of the base register Rn and the displacement field is used as the base address component of the bit address. The base address and the bit offset in register Rx are then combined to form the bit address.



mod Field Encoding

←mod→ m

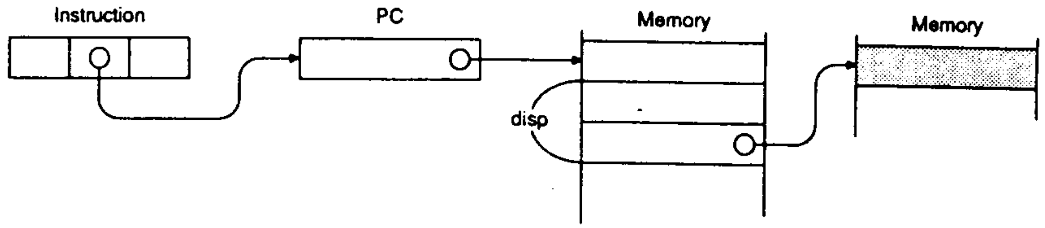
	47	31	23	15	7	0	m
[disp8[Rn]](Rx)			←disp8→	1 0 0 ←Rn→	1 1 0 ←Rx→		1
[disp16[Rn]](Rx)		←disp16→		1 0 1 ←Rn→	1 1 0 ←Rx→		1
[disp32[Rn]](Rx)	←disp32→			1 1 0 ←Rn→	1 1 0 ←Rx→		1

PC Displacement Indirect [disp[PC]]

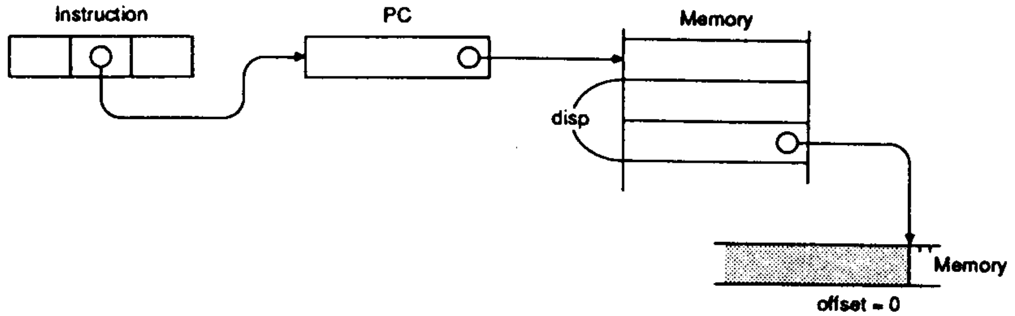
Format [disp[PC]].........Byte Addressing @[disp[PC]]................Bit Addressing

Description

Byte The word contents of the location addressed by the sum of the PC and the sign extended displacement field is used as the address of the operand.



Bit The word contents of the location addressed by the sum of the PC and the sign extended displacement field is used as the base address component for the bit address. There is no bit offset field specified by the addressing mode.



mod Field Encoding

←mod→ m

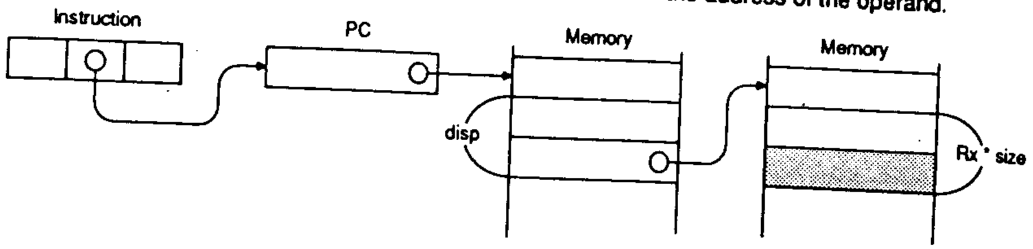
	39	23	15	7	0	m
[disp8[PC]]			←disp8→	1 1 1 1 1 0 0 0		0
[disp16[PC]]		←disp16→		1 1 1 1 1 0 0 1		0
[disp32[PC]]	←disp32→			1 1 1 1 1 0 1 0		0

PC Displacement Indirect Indexed [disp[PC]](Rx)

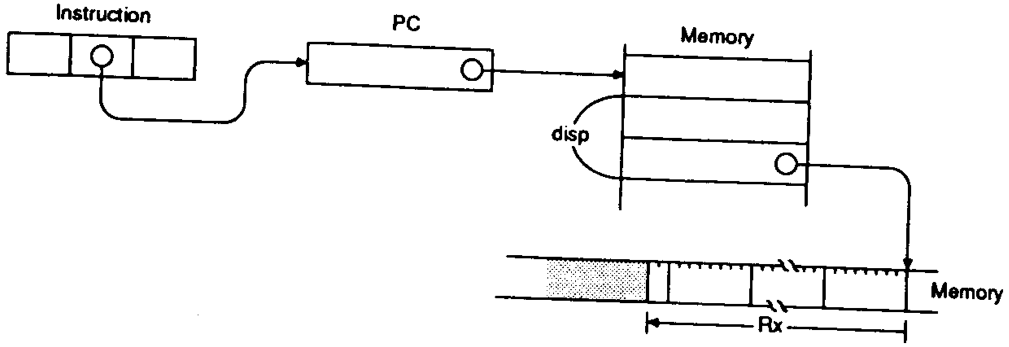
Format [disp[PC]](Rx).........Byte Addressing Rx@[disp[PC]]............Bit Addressing

Description

Byte The word contents of the location addressed by the sum of the PC and the sign extended displacement field is added to the scaled contents of the specified index register Rx to form the address of the operand.



Bit The word contents of the location addressed by the sum of the PC and the sign extended displacement field is treated as the byte component of the bit address and is combined with the bit offset in register Rx to form the bit address of the operand.



mod Field Encoding

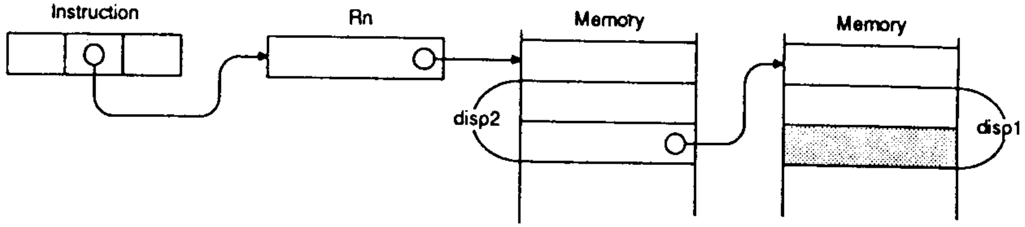
```
                    ◄──────────────────── mod ────────────────────►  m
                   47        31        23      15            7       0
[disp8[PC]](Rx)                        ◄─disp8─►1 1 1 1 1 0 0 0  1 1 0 ◄─Rx─►  1
[disp16[PC]](Rx)             ◄────disp16───────►1 1 1 1 1 0 0 1  1 1 0 ◄─Rx─►  1
[disp32[PC]](Rx) ◄─────────disp32──────────────►1 1 1 1 1 0 1 0  1 1 0 ◄─Rx─►  1
```

Double Displacement disp1[disp2[Rn]]

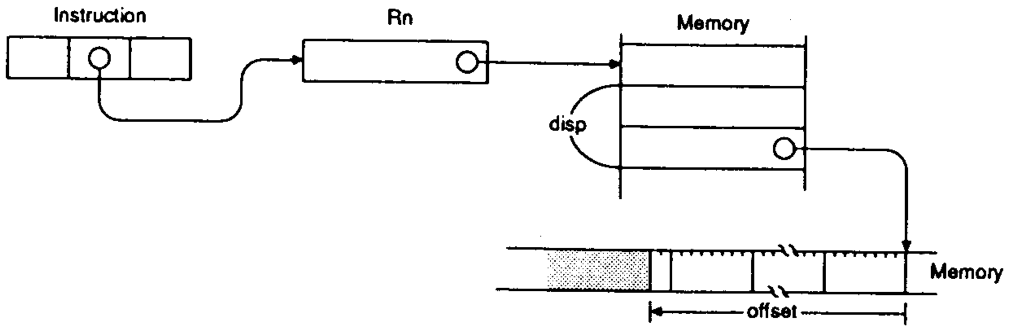
Format disp1[disp2[Rn]].....Byte Addressing offset@[disp[Rn]].........Bit Addressing

Description

Byte This addressing mode is like the displacement indirect mode except that an additional displacement value is provided to be added to the address of the operand. An 8-, 16-, or 32-bit displacement field, disp2, is sign extended and added to the contents of a general purpose register to form an address. The word contents of that location is then fetched and added to the first displacement field, disp1, to form the address of the operand.



Bit The word contents of the memory location addressed by the sum of the base register Rn and the displacement field is treated as the byte component of the bit address. This value is then combined with the bit offset field to generate the bit address of the operand.



mod Field Encoding

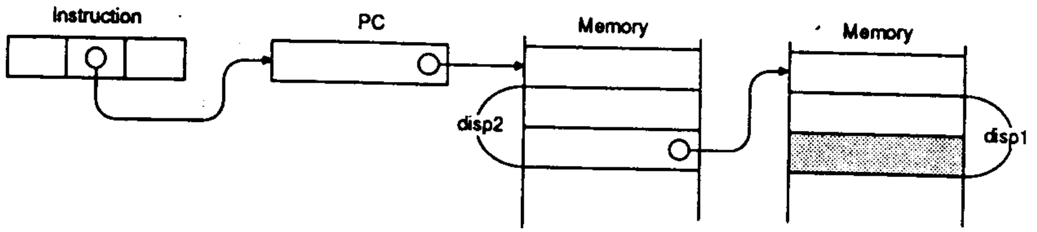
```
                                                   ◄──────────────── mod ────────────────►  m
                          71                       39        23     15      7             0
disp1.8[disp2.8[Rn]]                                         ◄disp1►◄disp2► 0 0 0  ◄─Rn─►  1
disp1.16[disp2.16[Rn]]                             ◄───disp1──►◄───disp2──► 0 0 1  ◄─Rn─►  1
disp1.32[disp2.32[Rn]] ◄──────disp1──────────────►◄──────────disp2────────► 0 1 0  ◄─Rn─►  1
```

PC Double Displacement disp1[disp2[PC]]

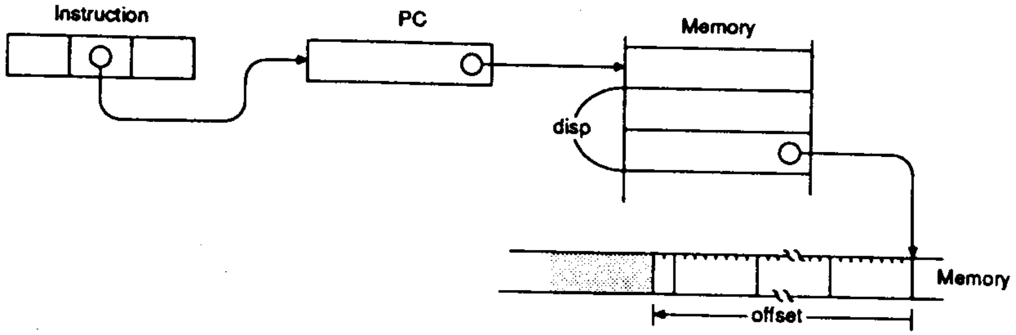
Format disp1[disp2[PC]].....Byte Addressing offset@[disp[PC]].........Bit Addressing

Description

Byte This addressing mode is similar to the displacement indirect mode except that an additional displacement value is provided to be added to the address of the operand. An 8-, 16-, or 32-bit displacement field, disp2, is sign extended and added to the contents of PC to form an address. The word contents of that address is then fetched and added to the first displacement field, disp1, to form the address of the operand.



Bit The word contents of the memory location addressed by the sum of the PC and the sign extended displacement field is treated as the byte component of the bit address. This value is then combined with the bit offset field to generate the bit address of the operand.



mod Field Encoding

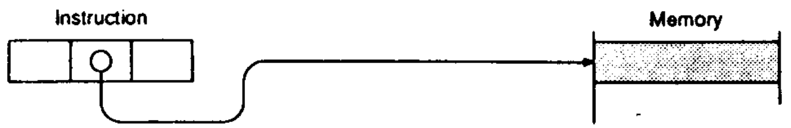
	71	39	23	15	7 0	m
		mod				m
disp1.8[disp2.8[Rn]]			disp1	disp2	1 1 1 1 1 1 0 0	0
disp1.16[disp2.16[Rn]]		disp1	disp2		1 1 1 1 1 1 0 1	0
disp1.32[disp2.32[Rn]]	disp1	disp2			1 1 1 1 1 1 1 0	0

Direct Address /addr

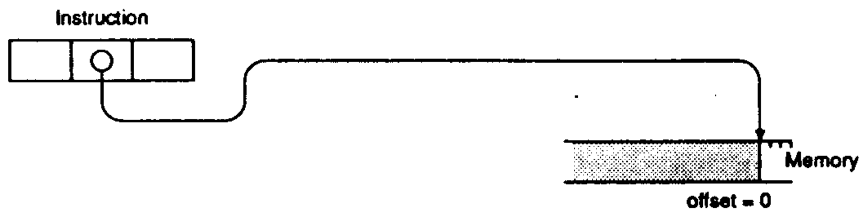
Format /addr..........Byte Addressing @/addr............Bit Addressing

Description

Byte The 32-bit address of the operand is explicitly contained in the instruction.



Bit The 32-bit address in the instruction is used as the byte address component of a bit address. The bit offset in this addressing mode is 0.



mod Field Encoding

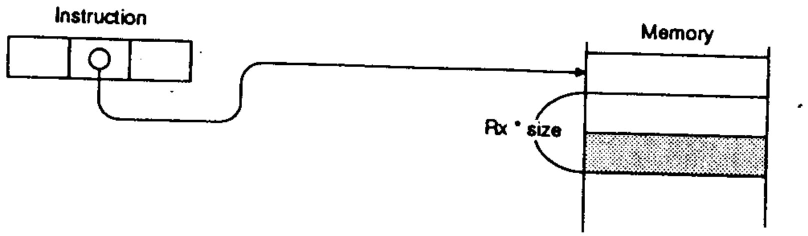
	39	23	15	7 0	m
	mod				m
/addr	addr.32			1 1 1 1 0 0 1 1	0

Direct Address Indexed /addr(Rx)

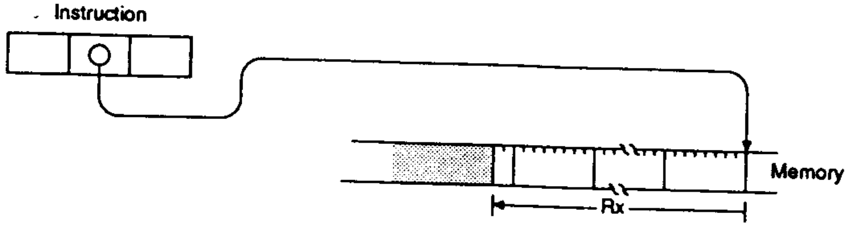
Format /addr(Rx)..........Byte Addressing Rx@/addr.........Bit Addressing

Description

Byte The sum of the operand address contained in the instruction and the scaled contents of index register Rx is used as the operand address.



Bit The 32-bit address in the instruction is treated as the byte component of a bit address and is combined with the bit offset located in register Rx to form the bit address of the operand.



mod Field Encoding

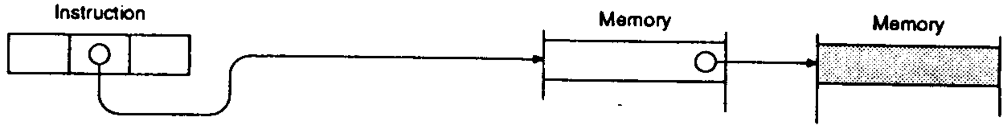
```
             ←─────────────────── mod ───────────────────→          m
             47        31        23        15          7        0
/addr(Rx)    ←──────── addr32 ──────────→ 1 1 1 1 0 0 1 1   1 1 0 ←Rx→   1
```

Direct Address Deferred [/addr]

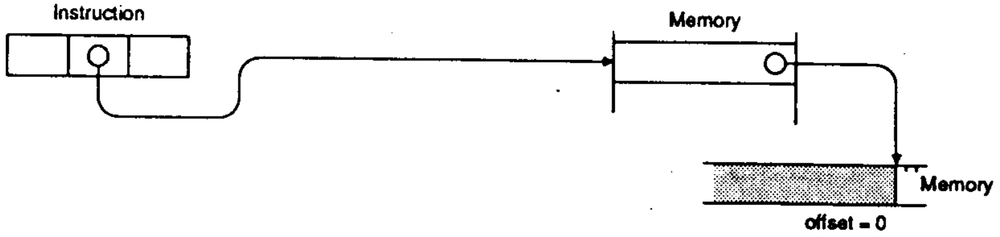
Format [/addr].............Byte Addressing @[/addr]..........Bit Addressing

Description

Byte As with the direct addressing mode, an address is specified explicitly in the instruction. The word contents of that location is then used as the address of the operand.



Bit The word contents of the location addressed by the instruction is treated as the byte component of the bit operand and is combined with a bit offset of 0 to address the operand.



mod Field Encoding

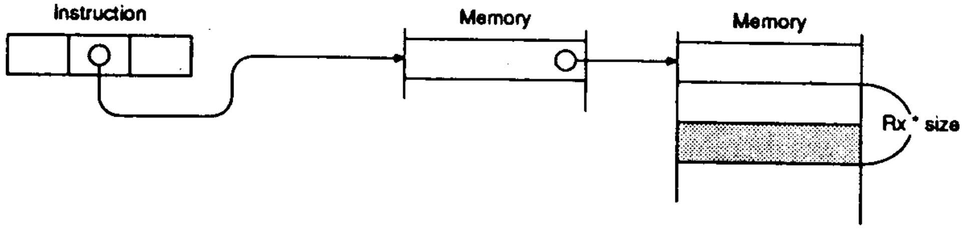
```
          ←──────────────── mod ────────────────→   m
          39        23        15        7        0
[/addr]   ←──────── addr.32 ─────────→ 1 1 1 1 1 0 1 1   0
```

Direct Address Deferred Indexed [/addr](Rx)

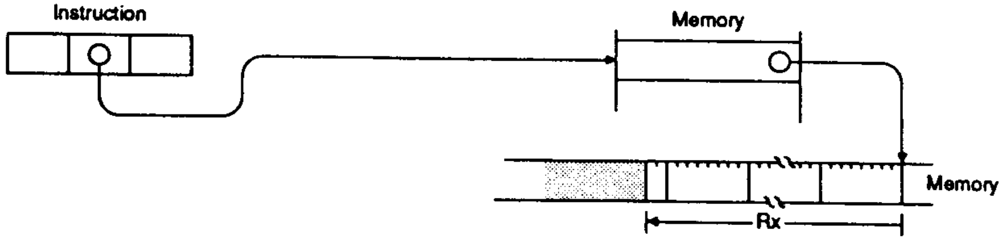
Format [/addr](Rx).............Byte Addressing Rx@[/addr].......Bit Addressing

Description

Byte The sum of the word contents of the memory location addressed by the instruction and the scaled contents of index register Rx is used as the address of the operand.



Bit The word contents of the location addressed by the instruction is treated as the byte component and combined with the bit offset located in register Rx to form the bit address of the operand.



mod Field Encoding

```
            ◄──────────────────── mod ────────────────────►     m
            47       31       23       15          7       0
[/addr](Rx) ◄──────── addr32 ──────────►1 1 1 1 1 0 1 1   1 1 0 ◄─Rx─►   1
```

Immediate #value

Format #value........Byte Addressing

Description

Byte With the immediate addressing mode, the operand is contained in the instruction. This addressing mode is useful for expressing constant values to be used as source data for instructions. The immediate addressing mode cannot be used with doubleword data.

Instruction

Bit Prohibited

mod Field Encoding

```
          ◄──────────────────── mod ────────────────────►   m
          39          23          15         7          0
#immed8                           ◄─immed8─►1 1 1 1 0 1 0 0   0
#immed16              ◄──── immed16 ───────►1 1 1 1 0 1 0 0   0
#immed32  ◄──────────── immed32 ───────────►1 1 1 1 0 1 0 0   0
```

Notes The use of the immediate mode as the destination operand addressing mode will result in a Illegal Addressing Mode exception.

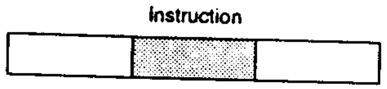
The attempted use of the immediate addressing mode as a doubleword source operand will result in a Reserved Addressing Mode exception.

Immediate Quick #value

Format #value..............Byte Addressing

Description

Byte This is a variant of the immediate addressing mode in which the immediate value is abbreviated to four bits and zero extended to the source operand length prior to instruction execution. The immediate quick addressing mode cannot be used with doubleword data.



Bit Prohibited

mod Field Encoding

```
          ←—— mod ——→     m
          7          0
#immed4   1 1 1 0←val→    0
```

Notes The use of the immediate quick mode as the destination operand addressing mode will result in a Illegal Addressing Mode exception.

The attempted use of the immediate addressing mode as a doubleword source operand will result in a Reserved Addressing Mode exception.

Addressing Mode Restrictions

Under certain conditions, restrictions are applied to the use of certain addressing modes in two operand instructions. Whenever the operand addressing modes are completely independent of each other, no restrictions apply to the selection and use of the µPD70616 addressing modes. However, if the two addressing modes have a mutual dependancy on a common base or index register, the operation may have unpredictable results.

Restrictions apply to two operand instructions when an autoincrement/autodecrement addressing mode is specified for the first operand and the second operand is addressed using a register modified by the first operand. Should this situation occur, the µPD70616 will calculate the effective addresses of the operands using the folowing two rules:

- the effective address of the first operand is evaluated and any modification are made

  then,

- the effective address of the second address is calculated and any modifications made.

For example, consider the following instruction.

```
mov.w    [ r3+ ], [ r3+ ]        ; OK
```

Using the preceding guidelines, if the contents of the `r3` is `0x100` prior to the execution of the instruction then the address of the first operand is `0x100`, the address of the second operand is `0x104` and the contents of `r3` following the execution of the instruction is `0x108`.

However, under the following conditions, the µPD70616 cannot reliably calculate the operand address and the restrictions below apply :

1. Autoincrement/autodecrement – scaled index

   If the autoincrement/autodecrement addressing mode is specified for the first operand and the same register is specified as the index register in a scaled index addressing mode for the second operand, the results are unpredictable.

   ```
   mov.w    [ r3+ ], [ r4 ]( r3 )   ; unpredictable
   ```

2. Autoincrement/autodecrement – autoincrement/autodecrement

   If the same register is specified for an autoincrement/autodecrement addressing mode for both the first and second operands of an instruction in which the lengths of the operand data types differ, the results are unpredictable.

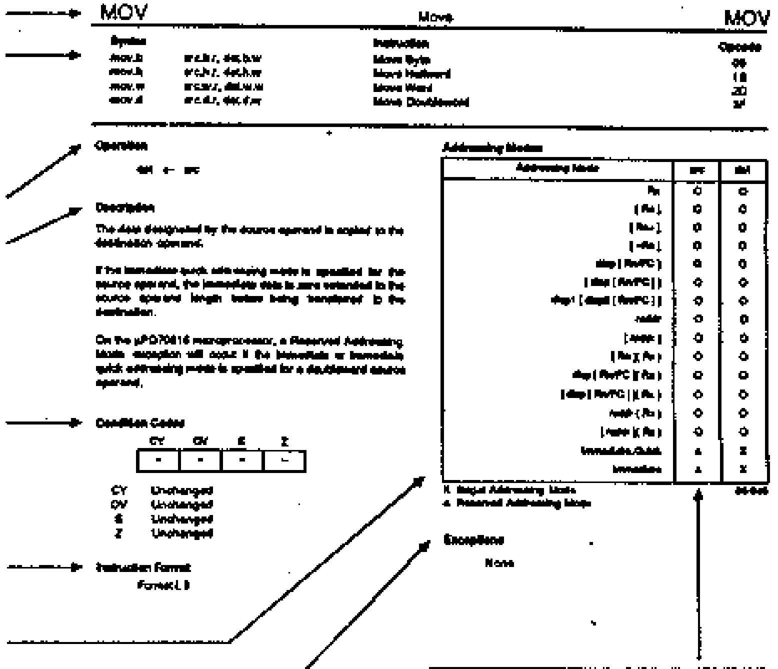
   ```
   movs.bw  [ -r20 ], [ -r20 ]      ; unpredictable
   cvt.lw   [ r20+ ], [ r20+ ]      ; unpredictable
   ```

Section 7
µPD70616 Instruction Set

A major goal of the µPD70616 design was to provide a comprehensive instruction set with full support for a variety of data types. A large variety of data transfer, arithmetic, logical, control transfer, and stack instructions are represented. The comprehensive instruction set coupled with the flexible addressing modes permit an optimizing compiler to extract maximum performance from the µPD70616 architecture, providing a performance increase over less sophisticated microprocessors. Additional instructions support system activities such as memory management and task swapping. These instructions are part of the privileged instruction set.

Instruction Set Description

Prior to introducing the instruction set descriptions, a number of commonly used definitions and an example should prove helpful.



Mnemonic

The assembler syntax for the instruction and any associated operands. The operand field contain three components, the operand name, data type and access fields. See the tables in the text for additional information on these fields.

A concise description of the instruction.

A detailed description of the instruction with additional information concerning the operation, effects on the flags and potential exception conditions.

This field contains information on the effect of the instruction execution on the flags. Any flags not listed are unmodified by the instruction.

The permissible instruction formats for this instruction.

A list of the permissible addressing modes for each operand.

A list of any exceptions that might occur as a result of the execution of this instruction. This list only includes the instruction exceptions and does not contain any of the memory management exceptions which can occur on any memory access.

Symbol Definitions

O Valid addressing mode
X Illegal addressing mode
Δ Reserved addressing mode
– Unavailable addressing mode

83-138

Data Type Abbreviations

Symbol	Data Type
b	Byte (8-bit)
h	Halfword (16-bit)
w	Word (32-bit)
d	Doubleword (64-bit)
s	Short Real (32-bit)
l	Long Real (64-bit)
p	Pointer (32-bit)

86-136

Operand Access Abbreviations

Symbol	Access Type
r	Read access
w	Write access
rw	Read and write access
rwi	Read-modify-write interlocked access
ex	Execute access
n	No access (MOVEA instruction)

86-135

The µPD70616 instruction set description is presented in alphabetical order. Appendix A provides a summary of the instruction set and also lists the instructions by functional grouping.

ABSF Absolute Value ABSF

Syntax		Instruction	Opcode
absf.s	src.s.r, dst.s.w	Absolute Value Short Real	5C•0A
absf.l	src.l.r, dst.l.w	Absolute Value Long Real	5E•0A

Operation

dst ← | src |

Description

The absolute value of the source operand is stored in the destination operand. Both the integer condition codes and the floating point condition codes are updated to reflect the result of the operation.

If the source operand is a NaN or an infinity, a Reserved Floating Point Operand exception will occur and the flags and destination will remain unchanged.

Condition Codes

CY	OV	S	Z
0	0	0	•

CY Cleared
OV Cleared
S Cleared
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
•	–	–	•	–

FIV Set if an invalid operation is attempted, otherwise unchanged
FZD Unchanged
FOV Unchanged
FUD Set if the result is denormal, otherwise unchanged
FPR Unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

86-045

Exceptions

Reserved Floating Point Operand
Floating Point Underflow

ADD — Add — ADD

Syntax		Instruction	Opcode
add.b	src.b.r, dst.b.rw	Add Byte	80
add.h	src.h.r, dst.h.rw	Add Halfword	82
add.w	src.w.r, dst.w.rw	Add Word	84

Operation

dst ← dst + src

Description

The sum of the source and destination operands is stored in the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
•	•	•	•

CY Set if a carry is generated, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[−Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode 86-046

Exceptions

None

ADDC — Add with Carry — ADDC

Syntax		Instruction	Opcode
addc.b	src.b.r, dst.b.rw	Add Byte with Carry	90
addc.h	src.h.r, dst.h.rw	Add Halfword with Carry	92
addc.w	src.w.r, dst.w.rw	Add Word with Carry	94

Operation

dst ← dst + src + CY

Description

The sum of the CY flag, source, and destination operands is stored in the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
•	•	•	•

CY Set if a carry is generated, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[−Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode 86-046

Exceptions

None

ADDDC Add Decimal with Carry ADDDC

Syntax		Instruction	Opcode
adddc	src.b.r, dst.b.rw, pat.b.r	Add Decimal with Carry	59•00

Operation

dst ← dst + src + CY using mask pattern

Description

The CY flag and the decimal source operand are added to the decimal destination operand and the result is stored in the destination operand. The decimal addition operation occurs only for the unmasked portion of the operands, as determined by the mask pattern.

The CY flag will be set if there is a carry out of the addition operation. If the result is non-zero or a carry is generated, the Z flag will be cleared, otherwise it remains unchanged.

Following the addition operation, the result is checked to verify that a valid BCD representation exists in the unmasked portion of the result. If either value is not a valid BCD digit (0-9), a Decimal Format exception will occur and the destination will remain unchanged.

Condition Codes

CY	OV	S	Z
•	–	–	•

CY Set if a carry is generated, otherwise cleared
OV Unchanged
S Unchanged
Z Unchanged if the result is zero, otherwise cleared

Instruction Format

Format VIIc

Addressing Modes

Addressing Mode	src	dst	pat
Rn	O	O	–
[Rn]	O	O	–
[Rn+]	O	O	–
[–Rn]	O	O	–
disp [Rn/PC]	O	O	–
[disp [Rn/PC]]	O	O	–
disp1 [disp2 [Rn/PC]]	O	O	–
/addr	O	O	–
[/addr]	O	O	–
[Rn](Rx)	O	O	–
disp [Rn/PC](Rx)	O	O	–
[disp [Rn/PC]](Rx)	O	O	–
/addr (Rx)	O	O	–
[/addr](Rx)	O	O	–
Immediate.Quick	O	X	–
Immediate	O	X	O

86-054

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

Decimal Format

ADDF Add Floating ADDF

Syntax		Instruction	Opcode
addf.s	src.s.r, dst.s.rw	Add Short Real	5C•18
addf.l	src.l.r, dst.l.rw	Add Long Real	5E•18

Operation

dst ← src + dst

Description

The sum of the source and destination operands is stored in the destination operand. Both the integer condition codes and the floating point condition codes are updated to reflect the result of the operation.

If the absolute values of the source and destination operands are equal but differ in sign, the sign of the zero result will be determined by the programmed rounding mode.

If a source or destination operand is a NaN or an infinity, a Reserved Floating Point Operand exception will occur and the flags and destination will remain unchanged.

Condition Codes

CY	OV	S	Z
•	0	•	•

CY Set if the result is negative and non-zero, otherwise cleared
OV Cleared
S Set if the mantissa sign bit of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
•	–	•	•	•

FIV Set if an invalid operation is attempted, otherwise unchanged
FZD Unchanged
FOV Set if the result is infinite, otherwise unchanged
FUD Set if the destination result is denormal, otherwise unchanged
FPR Set if a precision error occurs, otherwise unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

86-045

X Illegal Addressing Mode
Δ Reserved Addressing Mode

Exceptions

Reserved Floating Point Operand
Floating Point Overflow
Floating Point Underflow
Floating Point Precision

AND

Logical AND

AND

Syntax		Instruction	Opcode
and.b	src.b.r, dst.b.rw	AND Byte	A0
and.h	src.h.r, dst.h.rw	AND Halfword	A2
and.w	src.w.r, dst.w.rw	AND Word	A4

Operation

dst ← dst ∧ src

Description

The bit-wise AND of the source operand and destination operands is stored in the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	*	*

CY Unchanged
OV Cleared
S Set if the MSB of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode
86-046

Exceptions

None

ANDBS

AND Bit String

ANDBS

Syntax		Instruction	Opcode
andbsu	bsrc.b.r, blen.b.r, bdst.b.rw	AND Bit String (Upward)	5B•10
andbsd	bsrc.b.r, blen.b.r, bdst.b.rw	AND Bit String (Downward)	5B•11

Operation

bdst ← bsrc ∧ bdst

Description

The bit-wise AND of the source and destination bit strings is stored in the destination bit string. Specifying the direction of the operation allows the correct result to be computed, even when the source and destination bit strings overlap.

To minimize the interrupt latency time, the ANDBS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

During the execution of the ANDBS instruction, registers R28 and R27 contain pointers to the bytes within the source and destination bit strings to be processed next. Following the execution of the instruction, R28 contains the address of the byte containing the final bit of the source bit string while R27 contains the address of the byte containing the final bit of the destination bit string.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	bsrc	blen	bdst
Rn	X	O	X
@[Rn]	O	–	O
@[Rn+]	O	–	O
@[–Rn]	O	–	O
offset@[Rn/PC]	O	–	O
@[disp [Rn/PC]]	O	–	O
offset@[disp [Rn/PC]]	O	–	O
@/addr	O	–	O
@[/addr]	O	–	O
Rx@[Rn]	O	–	O
Rx@[Rn/PC]	O	–	O
Rx@[disp [Rn/PC]]	O	–	O
Rx@/addr	O	–	O
Rx@[/addr]	O	–	O
Immediate.Quick	X	–	X
Immediate	X	O	X

X Illegal Addressing Mode
– Unavailable Addressing Mode
86-052

Exceptions

None

ANDNBS AND Complemented Bit String ANDNBS

Syntax		Instruction	Opcode
andnbsu	bsrc.b.r, blen.b.r, bdst.b.rw	AND Complemented Bit String (Upward)	5B•12
andnbsd	bsrc.b.r, blen.b.r, bdst.b.rw	AND Complemented Bit String (Downward)	5B•13

Operation

```
bdst ← ~bsrc ∧ bdst
```

Description

The bit-wise AND of the complemented source bit string and the destination bit string is stored in the destination bit string. Specifying the direction of the operation allows the correct result to be computed, even when the source and destination bit strings overlap.

To minimize the interrupt latency time, the ANDNBS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

During the execution of the ANDNBS instruction, registers R28 and R27 contain pointers to the bytes within the source and destination bit strings to be processed next. Following the execution of the instruction, R28 contains the address of the byte containing the final bit of the source bit string while R27 contains the address of the byte containing the final bit of the destination bit string.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	bsrc	blen	bdst
Rn	X	O	X
@[Rn]	O	–	O
@[Rn+]	O	–	O
@[–Rn]	O	–	O
offset@[Rn/PC]	O	–	O
@[disp [Rn/PC]]	O	–	O
offset@[disp [Rn/PC]]	O	–	O
@/addr	O	–	O
@[/addr]	O	–	O
Rx@[Rn]	O	–	O
Rx@[Rn/PC]	O	–	O
Rx@[disp [Rn/PC]]	O	–	O
Rx@/addr	O	–	O
Rx@[/addr]	O	–	O
Immediate.Quick	X	–	X
Immediate	X	O	X

X Illegal Addressing Mode
– Unavailable Addressing Mode

86-052

Exceptions

None

Bcc Conditional Branch Bcc

Syntax		Instruction	Opcode
bcc	disp8	Branch on Condition (byte displacement)	6x
bcc	disp16	Branch on Condition (halfword displacement)	7x

Operation

```
if condition then
    PC ← PC + sign_extended( disp )
else
    PC ← NextPC
```

Description

The specified condition is tested and if true a branch is taken. The target address is computed by sign extending the 8- or 16-bit displacement field to 32 bit length and adding it to the PC.

The PC relative addressing mode is implicitedly selected by these instructions. The value of the PC used to compute the target address is the first byte of the branch instruction.

	Mnemonic	Condition
Signed	BGT	Branch if Greater
	BGE	Branch if Greater or Equal
	BLT	Branch if Less
	BLE	Branch if Less or Equal
Unsigned	BH	Branch if Higher
	BNL	Branch if Not Lower
	BL	Branch if Lower
	BNH	Branch if Hot Higher
Flags	BE	Branch if Equal
	BNE	Branch if Not Equal
	BV	Branch if Overflow
	BNV	Branch if No Overflow
	BN	Branch if Negative
	BP	Branch if Positive
	BC	Branch if Carry
	BNC	Branch if No Carry
	BZ	Branch if Zero
	BNZ	Branch if Not Zero
	BR	Unconditional Branch

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format IV

Mnemonic	Condition	Opcodes
BGT	$((S \oplus OV) \vee Z) = 0$	6F/7F
BGE	$(S \oplus OV) = 0$	6D/7D
BLT	$(S \oplus OV) = 1$	6C/7C
BLE	$((S \oplus OV) \vee Z) = 1$	6E/7E
BH	$(CY \vee Z) = 0$	67/77
BNL	CY = 0	63/73
BL	CY = 1	62/72
BNH	$(CY \vee Z) = 1$	66/76
BE	Z = 1	64/74
BNE	Z = 0	65/75
BV	OV = 1	60/70
BNV	OV = 0	61/71
BN	S = 1	68/78
BP	S = 0	69/79
BC	CY = 1	62/72
BNC	CY = 0	63/73
BZ	Z = 1	64/74
BNZ	Z = 0	65/75
BR	Always	6A/7A

Exceptions

None

BRK Break BRK

Syntax	Instruction	Opcode
brk	Breakpoint Trap	C8

Operation

[–SP] ← Exception Code
[–SP] ← PSW
[–SP] ← NextPC
PC ← [Exception Vector 13]

Description

The breakpoint trap is asserted and program control is transferred to the breakpoint trap exception handler.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format V

Exceptions

Breakpoint Trap

BRKV Break on Overflow BRKV

Syntax	Instruction	Opcode
brkv	Break on Overflow	C9

Operation

[–SP] ← CurrentPC
[–SP] ← Exception Code
[–SP] ← PSW
[–SP] ← NextPC
PC ← [Exception Vector 21]

Description

The OV flag is tested and if set, an Integer Overflow Exception occurs. Otherwise, instruction execution continues witht the next instruction.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format V

Exceptions

Integer Overflow

BSR Branch to Subroutine BSR

Syntax		Instruction	Opcode
bsr	disp16	Branch to Subroutine	48

Operation

```
[-SP] ← NextPC
PC ← PC + sign_extended( disp16 )
```

Description

The address of the next instruction is pushed onto the stack and control is transferred to the address computed by adding the sign extended16-bit displacment to the PC.

The PC relative addressing mode is implicitedly selected by this instructions. The value of the PC used to compute the target address is the first byte of the branch instruction.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format IV

Exceptions

None

CALL Call Procedure CALL

Syntax		Instruction	Opcode
call	target.b.ex, arg.w.r	Call Procedure	49

Operation

```
tmp1 ← effective_address( target )
tmp2 ← effective_address( arg )
[-SP] ← AP
AP ← tmp2
[-SP] ← NextPC
PC ← tmp1
```

Description

The CALL instruction is high level language oriented instruction for transferring control to a subordinate procedure.

The instruction operates by first calculating the effective addresses of the target procedure and argument list and saving the current AP and PC registers on the stack. The newly calculated values then replace the contents of the AP and PC registers.

When the autoincrement, autodecrement, or scaled index addressing modes are used for either the target or argument operands, the contents of the pointer are modified by four. Also, the result is unpredictable if the autoincrement or autodecrement addressing mode is specified using SP as the base register.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	target	arg
Rn	X	X
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	X	X
Immediate	X	X

X Illegal Addressing Mode

86-073

Exceptions

None

CAXI — Compare and Exchange Interlocked — CAXI

Syntax		Instruction	Opcode
caxi	Rn.w.rw, dst.w.rwi	Compare and Exchange Interlocked	4C

Operation

```
lock
flags ← dst – Rn
if ( Z = 1 ) then
    dst ← R28
else
    Rn ← dst
unlock
```

Description

This instruction is used to synchronize processes or provide mutual exclusion in a multiple processor configuration. CAXI is a more general form of the TASI instruction.

The processor informs other bus masters in the system that an indivisible operation will take place by asserting the bus lock output signal. The destination operand is then fetched and compared with Rn and if equal, the contents of R28 are stored in the destination. Otherwise the destination contents are placed in Rn. The bus lock output is then negated, indicating that other bus masters may again access the shared data.

If the register addressing mode is specified for the destination, the execution of the instruction is meaningless but the operation is carried out.

This instruction is not allowed to use Format II and furthermore, the Format I direction field must be zero.

Instruction Format

Format I

Condition Codes

CY	OV	S	Z
*	*	*	*

CY Set if a borrow is generated, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the results are negative, otherwise cleared
Z Set if the results are zero, otherwise cleared

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	X	O
[Rn+]	X	O
[–Rn]	X	O
disp [Rn/PC]	X	O
[disp [Rn/PC]]	X	O
disp1 [disp2 [Rn/PC]]	X	O
/addr	X	O
[/addr]	X	O
[Rn](Rx)	X	O
disp [Rn/PC](Rx)	X	O
[disp [Rn/PC]](Rx)	X	O
/addr (Rx)	X	O
[/addr](Rx)	X	O
Immediate.Quick	X	X
Immediate	X	X

X Illegal Addressing Mode

86-093

Exceptions

None

CHKA — Check Access Permission — CHKA

Syntax		Instruction	Opcode
chkar	va.p.r, level.b.r	Check Read Access Permission	4D
chkaw	va.p.r, level.b.r	Check Write Access Permission	4E
chkae	va.p.r, level.b.r	Check Execute Access Permission	4F

Operation

check memory access permissions

Description

A check is made if the byte data addressed by the virtual address can be accessed at the specified execution level. The Z flag will be set if the specified access is permitted. The CY flag indicates whether the virtual address is mapped into the I/O address space. The S flag will be set if the MMU was unable to complete the address translation.

An Illegal Data Field exception will occur if the execution level operand is not in the range

$$0 \le \text{level} \le 3$$

or the current execution level is less privileged than the level operand

$$\text{level} < \text{PSW.EL}$$

The absence of the area or page tables will cause a memory management fault just as in a normal data access, however, the page need not be physical present for access rights to be checked.

When executed in the real mode, the Z flag will be set and the CY and S flags cleared.

Condition Codes

CY	OV	S	Z
*	–	*	*

CY Set if the page is I/O mapped, otherwise cleared
OV Unchanged
S Set if the translation fails, otherwise cleared
Z Set if operation is permissible, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	va	level
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	O
Immediate	O	O

86-091

Exceptions

Illegal Data Field

CHLVL Change Level CHLVL

Syntax		Instruction	Opcode
chlvl	level.b.r, arg.b.r	Change Execution Level	4B

Operation

```
[-SP] ← zero_extended( arg )
[-SP] ← Exception Code
[-SP] ← PSW
[-SP] ← NextPC
PC ← [ Exception Vector ( 24 + level ) ]
```

Description

This instruction provides a protected method of accessing more privileged execution levels.

The execution level is changed to the new level and the byte argument is zero extended to word length pushed on the target execution level stack. The change execution level exception processing then pushes the exception code, PSW and PC of the next instruction on the stack and transfers control to the appropriate exception handler.

An Illegal Data Field exception will occur if the level operand is not in the range

$$0 \leq \text{level} \leq 3$$

or the current execution level is less than the level operand

$$\text{level} < \text{PSW.EL}$$

Operands are zero extended to byte length if the immediate quick addressing mode is specified.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	level	arg
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	O
Immediate	O	O

86-092

Exceptions

Illegal Data Field

CLR1 Clear Bit CLR1

Syntax		Instruction	Opcode
clr1	offset.w.r, base.w.rw	Bit Test and Clear	A7

Operation

```
CY ← bit( base, offset )
Z ← ~bit( base, offset )
bit( base, offset ) ← 0
```

Description

The bit located at the sum of the byte base address and bit offset is tested and then cleared. The CY and Z flags reflect the state of the bit prior to the execution of the instruction.

The location of the designated bit is determined by the base operand. If the register addressing mode is used for the base operand, the designated bit is located within a general purpose register at the specified bit offset. For any other addressing mode, the designated bit is in memory at the specified bit offset from the base address. An Illegal Data Field exception occurs if the bit offset is outside the range 0 to 31.

If the autoincrement or autodecrement addressing mode is specified for the base operand, the base operand is treated as word data and is incremented or decremented by four. When the immediate quick addressing mode is specified, the immediate data is zero extended to word length and used as the bit offset.

Condition Codes

CY	OV	S	Z
•	–	–	•

CY Set if the designated bit is 1, otherwise cleared
OV Unchanged
S Unchanged
Z Set if the designated bit is 0, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	offset	base
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode 86-095

Exceptions

Illegal Data Field

CLRTLB Clear TLB Entry CLRTLB

Syntax		Instruction	Opcode
clrtlb	va.p.r	Clear TLB Entry	FE/F

Operation

TLB Entry(va) ← Invalid

Description

The virtual address and the section designator (R28) are used to identify a TLB entry to be marked as invalid. The CLRTLB instruction only clears a TLB entry that has a matching virtual to physical address translation.

If the contents of R28 are 0FFFFFFFFH, the virtual address operand is translated using the current virtual address space. Otherwise, R28 is assumed to contain an area table base address and the virtual address is translated by ignoring the lower 3 bits of the area table base register and performing no area table length checking.

If the immediate quick addressing mode is specified for the virtual address operand, the immediate data is zero extended to 32-bit length and used as the virtual address.

This instruction can be executed in either the real or virtual address mode.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	va
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	O
Immediate	O

86-077

Exceptions

Privileged Instruction

CLRTLBA Clear All TLB Entries CLRTLBA

Syntax	Instruction	Opcode
clrtlba	Clear All TLB Entries	10

Operation

All TLB Entries ← Invalid

Description

Each translation lookaside buffer entry is voided.

This instruction can be executed in either the real or the virtual address mode.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format V

Exceptions

Privileged Instruction

CMP Compare CMP

Syntax		Instruction	Opcode
cmp.b	src1.b.r, src2.b.r	Compare Byte	B8
cmp.h	src1.h.r, src2.h.r	Compare Halfword	BA
cmp.w	src1.w.r, src2.w.r	Compare Word	BC

Operation

src2 – src1

Description

The source operands are compared by subtracting the first source operand from the second source operand and updating the flags in the PSW.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
*	*	*	*

CY Set if a borrow is generated, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src1	src2
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	O
Immediate	O	O

86-062

Exceptions

None

CMPBF Compare Bit Field CMPBF

Syntax		Instruction	Opcode
cmpbfs	bsrc.w.r, blen.b.r, src.w.r	Compare Sign Extended Bit Field	5D•00
cmpbfz	bsrc.w.r, blen.b.r, src.w.r	Compare Zero Extended Bit Field	5D•01
cmpbfl	bsrc.w.r, blen.b.r, src.w.r	Compare Left Justified Bit Field	5D•02

Operation

flags ← src – bitfield

Description

The designated bit field is extracted using the specified mode and compared to the source operand. The comparison is made by subtracting the bit field data from the word length source operand and storing the result in the condition codes.

If the bit field length is zero, zero will be subtracted from the source operand.

The sum of the bit offset and the bit field length must not exceed thirty-two, otherwise an Illegal Data Field exception will occur.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the word length before performing the comparison operation.

Condition Codes

CY	OV	S	Z
*	*	*	*

CY Set if a borrow is generated, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format VIIb

Addressing Modes

Bit Address	bsrc	blen	src	Byte Address
Rn	X	O	O	Rn
@[Rn]	O	–	O	[Rn]
@[Rn+]	O	–	O	[Rn+]
@[–Rn]	O	–	O	[–Rn]
offset@[Rn]	O	–	O	disp[Rn]
offset@[PC]	O	–	O	disp[PC]
@[disp[Rn]]	O	–	O	[disp[Rn]]
@[disp[PC]]	O	–	O	[disp[PC]]
offset@[disp[Rn]]	O	–	O	disp1[disp2[Rn]]
offset@[disp[PC]]	O	–	O	disp1[disp2[PC]]
@/addr	O	–	O	/addr
@[/addr]	O	–	O	[/addr]
Rx@[Rn]	O	–	O	[Rn](Rx)
Rx@[Rn]	O	–	O	disp [Rn](Rx)
Rx@[PC]	O	–	O	disp [PC](Rx)
Rx@[disp[Rn]]	O	–	O	[disp[Rn]](Rx)
Rx@[disp[PC]]	O	–	O	[disp[PC]](Rx)
Rx@/addr	O	–	O	/addr (Rx)
Rx@[/addr]	O	–	O	[/addr](Rx)
Immediate.Quick	X	–	O	Immediate.Quick
Immediate	X	O	O	Immediate

X Illegal Addressing Mode
– Unavailable Addressing Mode

86-061

Exceptions

Illegal Data Field

CMPC Compare Character CMPC

Syntax		Instruction	Opcode
cmpc.b	src.b.r, slen.b.r, dst.b.r, dlen.b.r	Compare Byte Character String	58•00
cmpc.h	src.h.r, slen.b.r, dst.h.r, dlen.b.r	Compare Halfword Character String	5A•00

Operation

flags ← dst – src

Description

The character string designated by the source operand is compared to the character string designated by the destination operand. The comparison continues until the end of either character string is reached or there is a disagreement between the string contents. Following the execution of the instruction, the S and Z flags are updated to reflect the relationship between the character strings.

The S flag reflects the lexical ordering of the character strings. If the compare instruction terminates with different characters, then the S flag reflects the unsigned comparison of the two strings. If the compare instruction terminates by reaching the of either string, the S flag will indicate the shorter string. The Z flag will be set if and only if the character strings are of identical length and contents.

During the comparison operation, registers R28 and R27 are used to maintain the source and destination addresses respectively. Following the execution of the CMPC instruction, these registers contain the addresses of the characters immediately following the the strings if the end of either string was reached. Otherwise, R28 and R27 will contain the addresses of the characters in disagreement.

To minimize the interrupt latency time, the CMPC instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

Condition Codes

CY	OV	S	Z
–	–	•	•

CY	Unchanged
OV	Unchanged
S	Set if src > dst, otherwise cleared
Z	Set if src = dst, otherwise cleared

Instruction Format

Format VIIa

Addressing Modes

Addressing Mode	src	slen	dst	dlen
Rn	X	O	X	O
[Rn]	O	–	O	–
[Rn+]	O	–	O	–
[–Rn]	O	–	O	–
disp [Rn/PC]	O	–	O	–
[disp [Rn/PC]]	O	–	O	–
disp1 [disp2 [Rn/PC]]	O	–	O	–
/addr	O	–	O	–
[/addr]	O	–	O	–
[Rn](Rx)	O	–	O	–
disp [Rn/PC](Rx)	O	–	O	–
[disp [Rn/PC]](Rx)	O	–	O	–
/addr (Rx)	O	–	O	–
[/addr](Rx)	O	–	O	–
Immediate.Quick	X	–	X	–
Immediate	X	O	X	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

88-048

Exceptions

Illegal Data Field

CMPCF Compare Character with Filler CMPCF

Syntax		Instruction	Opcode
cmpcf.b	src.b.r, slen.b.r, dst.b.r, dlen.b.r	Compare Byte Character String with Filler	58•01
cmpcf.h	src.h.r, slen.b.r, dst.h.r, dlen.b.r	Compare Halfword Character String with Filler	5A•01

Operation

flags ← dst – src

Description

The character string designated by the source operand is compared to the character string designated by the destination operand. The comparison operation continues until a disagreement between the string contents is detected or both strings are exhausted. If the source and destination character strings are not of equal length, the shorter string will be automatically extended using the fill character in R26 to the longer string length. Following the execution of the instruction, the S and Z flags are updated to reflect the relationship between the character strings.

The S flag reflects the lexical ordering of the character strings. If the compare instruction terminates with different characters, then the S flag reflects the unsigned comparison of the two strings. If the compare instruction terminates by reaching the of either string, the S flag will indicate the shorter string. The Z flag will be set if and only if the character strings are of identical length and contents.

During the comparison operation, registers R28 and R27 are used to maintain the source and destination addresses respectively. Following the execution of the CMPCF instruction, these registers contain the addresses of the characters immediately following the the strings if the end of either string was reached. Otherwise, R28 and R27 will contain the addresses of the characters in disagreement.

To minimize the interrupt latency time, the CMPCF instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

Condition Codes

CY	OV	S	Z
–	–	•	•

CY	Unchanged
OV	Unchanged
S	Set if src > dst, otherwise cleared
Z	Set if src = dst, otherwise cleared

Instruction Format

Format VIIa

Addressing Modes

Addressing Mode	src	slen	dst	dlen
Rn	X	O	X	O
[Rn]	O	–	O	–
[Rn+]	O	–	O	–
[–Rn]	O	–	O	–
disp [Rn/PC]	O	–	O	–
[disp [Rn/PC]]	O	–	O	–
disp1 [disp2 [Rn/PC]]	O	–	O	–
/addr	O	–	O	–
[/addr]	O	–	O	–
[Rn](Rx)	O	–	O	–
disp [Rn/PC](Rx)	O	–	O	–
[disp [Rn/PC]](Rx)	O	–	O	–
/addr (Rx)	O	–	O	–
[/addr](Rx)	O	–	O	–
Immediate.Quick	X	–	X	–
Immediate	X	O	X	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

88-048

Exceptions

Illegal Data Field

CMPCS Compare Character with Stopper CMPCS

Syntax		Instruction	Opcode
cmpcs.b	src.b.r, slen.b.r, dst.b.r, dlen.b.r	Compare Byte Character String with Stopper	58•02
cmpcs.h	src.h.r, slen.b.r, dst.h.r, dlen.b.r	Compare Halfword Character String with Stopper	5A•02

Operation

flags ← dst − src

Description

The character string designated by the source operand is compared to the character string designated by the destination operand. The comparison operation continues until a disagreement between the string contents is detected a string is exhausted or the stop character in R26 is detected in either string. Following the execution of this instruction, the S, Z, and CY flags are updated to reflect the relationship between the character strings.

The S flag reflects the lexical ordering of the character strings. If the compare instruction terminates with different characters, then the S flag reflects the unsigned comparison of the two strings. If the compare instruction terminates by reaching the end of either string without detecting the stop character, the S flag will indicate the shorter string. The Z flag will be set if and only if the character strings are of identical length and content. The CY flag is cleared if the stop character is detected in either string, otherwise it is set.

During the comparison operation, registers R28 and R27 are used to maintain the source and destination addresses respectively. Following the execution of the CMPCS instruction, these registers contain the addresses of the characters immediately following the the strings if the end of either string was reached. Otherwise, R28 and R27 will contain the addresses of the characters in disagreement.

To minimize the interrupt latency time, the CMPCS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

Condition Codes

CY	OV	S	Z
*	−	*	*

CY Set if the compare operation terminates without detecting the stop character in either string, otherwise cleared
OV Unchanged
S Set if src > dst, otherwise cleared
Z Set if src = dst, otherwise cleared

Instruction Format

Format VIIa

Addressing Modes

Addressing Mode	src	slen	dst	dlen
Rn	X	O	X	O
[Rn]	O	–	O	–
[Rn+]	O	–	O	–
[–Rn]	O	–	O	–
disp [Rn/PC]	O	–	O	–
[disp [Rn/PC]]	O	–	O	–
disp1 [disp2 [Rn/PC]]	O	–	O	–
/addr	O	–	O	–
[/addr]	O	–	O	–
[Rn](Rx)	O	–	O	–
disp [Rn/PC](Rx)	O	–	O	–
[disp [Rn/PC]](Rx)	O	–	O	–
/addr (Rx)	O	–	O	–
[/addr](Rx)	O	–	O	–
Immediate.Quick	X	–	X	–
Immediate	X	O	X	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

86-048

Exceptions

Illegal Data Field

CMPF Compare Floating CMPF

Syntax		Instruction	Opcode
cmpf.s	src1.s.r, src2.s.r	Compare Short Real	5C•00
cmpf.l	src1.l.r, src2.l.r	Compare Long Real	5E•00

Operation

Flags ← src2 − src1

Description

The difference of the two source operands is computed and the integer and floating point condition codes are updated to reflect the result of the operation.

If either source operand is a NaN or an infinity, a Reserved Floating Point Operand exception will occur and the flags will remain unmodified.

Condition Codes

CY	OV	S	Z
*	*	*	*

CY Set if the result is negative, otherwise cleared
OV Set if unordered, otherwise cleared
S Set to the MSB of the result
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
*	–	–	–	–

FIV Set if an invalid operation is attempted, otherwise unchanged
FZD Unchanged
FOV Unchanged
FUD Unchanged
FPR Unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src1	src2
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	Δ
Immediate	Δ	Δ

Δ Reserved Addressing Mode

86-068

Exceptions

Reserved Floating Point Operand
Invalid Floating Point Operation

CVT Convert CVT

Syntax		Instruction	Opcode
cvt.sl	src.s.r, dst.l.w	Convert Short Real to Long Real	5F•10
cvt.ls	src.l.r, dst.s.w	Convert Long Real to Short Real	5F•08

Operation

dst ← src

Description

The source operand is converted to the destination operand format. The integer and floating point condition codes are updated to reflect the result of the operation.

Condition Codes

CY	OV	S	Z
•	0	•	•

CY Set if the result is negative and non-zero, otherwise cleared
OV Cleared
S Set if the mantissa sign bit of the result is set, otherwise cleared
Z Set if the destination is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
•	–	•	•	•

FIV Set if an invalid operation is attempted, otherwise unchanged
FZD Unchanged
FOV Set if the result is infinite, otherwise unchanged
FUD Set if the destination result is denormal, otherwise unchanged
FPR Set if a precision error occurs, otherwise unchanged

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

84-045

Exceptions

Reserved Floating Point Operand
Floating Point Overflow
Floating Point Underflow
Floating Point Precision

Instruction Format

Format II

CVT Convert CVT

Syntax		Instruction	Opcode
cvt.ws	src.w.r, dst.l.w	Convert Word to Short Real	5F•00
cvt.wl	src.w.r, dst.l.w	Convert Word to Long Real	5F•11

Operation

dst ← src

Description

The word source operand is converted to the destination operand format. The integer and floating point condition codes are updated to reflect the result of the operation.

Condition Codes

CY	OV	S	Z
•	0	•	•

CY Set if the result is negative and non-zero, otherwise cleared
OV Cleared
S Set if the mantissa sign bit of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
–	–	–	–	•

FIV Unchanged
FZD Unchanged
FOV Unchanged
FUD Unchanged
FPR Set if a precision error occurs, otherwise unchanged

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

84-045

Exceptions

Floating Point Precision

Instruction Format

Format II

CVT — Convert — CVT

Syntax		Instruction	Opcode
cvt.sw	src.s.r, dst.w.w	Convert Short Real to Word	5F•01
cvt.lw	src.l.r, dst.w.w	Convert Long Real to Word	5F•09

Operation

dst ← src

Description

The source operand is converted to the word data type. The integer and floating point condition codes are updated to reflect the result of the operation.

Condition Codes

CY	OV	S	Z
–	•	•	•

CY Unchanged
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
•	–	–	–	•

FIV Set if an invalid operation is attempted, otherwise unchanged
FZD Unchanged
FOV Unchanged
FUD Unchanged
FPR Set if a precision error occurs, otherwise unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

86-045

X Illegal Addressing Mode
Δ Reserved Addressing Mode

Exceptions

Reserved Floating Point Operand
Invalid Floating Point Operation
Floating Point Precision

CVTD.PZ — Convert Decimal — CVTD.PZ

Syntax		Instruction	Opcode
cvtd.pz	src.b.r, dst.h.w, pat.b.r	Convert Packed to Zoned Decimal	59•10

Operation

```
tmp[3:0]     ←  src[7:4]
tmp[7:4]     ←  0
tmp[11:8]    ←  src[3:0]
tmp[15:12]   ←  0
dst[7:0]     ←  tmp[7:0] v pat[7:0]
dst[15:8]    ←  tmp[15:8] v pat[7:0]
if src = 0 then
    Z ← Z
else
    Z ← 0
```

Description

The byte length source operand is unpacked by performing a bit-wise OR or the digits with the pattern operand.

Prior to the conversion, the source operand is checked to verify a that a valid BCD representation exists in the unmasked portion of the data. If either value is not a legal BCD digit (0-9), a Decimal Format exception will occur and the destination will remain unchanged.

Condition Codes

CY	OV	S	Z
–	–	–	•

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged if the source operand is zero, otherwise cleared

Instruction Format

Format VIIc

Addressing Modes

Addressing Mode	src	dst	pat
Rn	O	O	–
[Rn]	O	O	–
[Rn+]	O	O	–
[–Rn]	O	O	–
disp [Rn/PC]	O	O	–
[disp [Rn/PC]]	O	O	–
disp1 [disp2 [Rn/PC]]	O	O	–
/addr	O	O	–
[/addr]	O	O	–
[Rn](Rx)	O	O	–
disp [Rn/PC](Rx)	O	O	–
[disp [Rn/PC]](Rx)	O	O	–
/addr (Rx)	O	O	–
[/addr](Rx)	O	O	–
Immediate.Quick	O	X	–
Immediate	O	X	O

86-054

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

Decimal Format

CVTD.ZP Convert Decimal CVTD.ZP

Syntax		Instruction	Opcode
cvtd.zp	src.h.r, dst.b.w, pat.b.r	Convert Zoned to Packed Decimal	59•18

Operation

```
if ( src[7:4] ≠ pat[7:4] ) or
            ( src[15:12] ≠ pat[7:4] ) then
    Decimal_Format_Exception
if ( src[3:0] > 9 ) or ( src[11:8] > 9 ) then
    Decimal_Format_Exception
dst[3:0] ← src[11:8]
dst[7:4] ← src[3:0]
if dst = 0 then
    Z ← Z
else
    Z ← 0
```

Description

The halfword source operand is converted from zoned decimal format to packed decimal format and stored in the destination operand.

Prior to the conversion, the source operand is checked to verify that a valid BCD representation exists in the lower nibbles of the upper and lower bytes. The upper nibbles are then compared to the upper nibble of the mask pattern. If either condition exists, a Decimal Format exception will occur and the destination will remain unchanged.

Condition Codes

CY	OV	S	Z
–	–	–	•

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged if the destination is zero, otherwise cleared

Instruction Format

Format VIIc

Addressing Modes

Addressing Mode	src	dst	pat
Rn	O	O	–
[Rn]	O	O	–
[Rn+]	O	O	–
[–Rn]	O	O	–
disp [Rn/PC]	O	O	–
[disp [Rn/PC]]	O	O	–
disp1 [disp2 [Rn/PC]]	O	O	–
/addr	O	O	–
[/addr]	O	O	–
[Rn](Rx)	O	O	–
disp [Rn/PC](Rx)	O	O	–
[disp [Rn/PC]](Rx)	O	O	–
/addr (Rx)	O	O	–
[/addr](Rx)	O	O	–
Immediate.Quick	O	X	–
Immediate	O	X	O

86-054

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

Decimal Format

DBcc Decrement and Conditionally Branch DBcc

Syntax		Instruction	Opcode
dbcc	Rn.w.rw, disp16	Decrement and Branch on Condition	C6•x/C7•x

Operation

```
Rn ← Rn – 1
if ( condition and Rn ≠ 0 ) then
    PC ← PC + sign_extended( disp16 )
else
    PC ← NextPC
```

Description

The specified general purpose register is decremented and if not zero and the specified condition is met, the branch is taken.

The PC relative addressing mode is implicitedly selected by these instructions. The 16-bit displacement field is sign extended to 32 bit length and added to the PC to compute the target address. The value of the PC used to compute the target address is the first byte of the decrement and branch instruction.

	Mnemonic	Condition
Signed	DBGT	Branch if Greater
	DBGE	Branch if Greater or Equal
	DBLT	Branch if Less
	DBLE	Branch if Less or Equal
Unsigned	DBH	Branch if Higher
	DBNL	Branch if Not Lower
	DBL	Branch if Lower
	DBNH	Branch if Hot Higher
Flags	DBE	Branch if Equal
	DBNE	Branch if Not Equal
	DBV	Branch if Overflow
	DBNV	Branch if No Overflow
	DBN	Branch if Negative
	DBP	Branch if Positive
	DBC	Branch if Carry
	DBNC	Branch if No Carry
	DBZ	Branch if Zero
	DBNZ	Branch if Not Zero
	DBR	Unconditional Branch

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VI

Mnemonic	Condition	Opcode
DBGT	$((S \oplus OV) \vee Z) = 0$	C7•7
DBGE	$(S \oplus OV) = 0$	C7•6
DBLT	$(S \oplus OV) = 1$	C6•6
DBLE	$((S \oplus OV) \vee Z) = 1$	C8•7
DBH	$(CY \vee Z) = 0$	C7•3
DBNL	CY = 0	C7•1
DBL	CY = 1	C6•1
DBNH	$(CY \vee Z) = 1$	C6•3
DBE	Z = 1	C6•2
DBNE	Z = 0	C7•2
DBV	OV = 1	C6•0
DBNV	OV = 0	C7•0
DBN	S = 1	C6•4
DBP	S = 0	C7•4
DBC	CY = 1	C6•1
DBNC	CY = 0	C7•1
DBZ	Z = 1	C6•2
DBNZ	Z = 0	C7•2
DBR	Always	C6•5

Exceptions

None

DEC Decrement DEC

Syntax		Instruction	Opcode
dec.b	dst.b.rw	Decrement Byte	D0/1
dec.h	dst.h.rw	Decrement Halfword	D2/3
dec.w	dst.w.rw	Decrement Word	D4/5

Operation

dst ← dst – 1

Description

The contents of the destination operand are decremented.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

The DEC instruction is a shorter encoding for the more general instruction

sub #1, dst

Condition Codes

CY	OV	S	Z
*	*	*	*

CY	Set if a borrow is generated, otherwise cleared
OV	Set if integer overflow occurs, otherwise cleared
S	Set if the result is negative, otherwise cleared
Z	Set if the result is zero, otherwise cleared

Instruction Format

Format III

Addressing Modes

Addressing Mode	dst
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	X
Immediate	X

X Illegal Addressing Mode 83-058

Exceptions

None

DISPOSE Dispose DISPOSE

Syntax	Instruction	Opcode
dispose	Dispose Stack Frame	CC

Operation

SP ← FP
FP ← [SP+]

Description

The DISPOSE instruction deletes the current stack frame by copying the contents of the frame pointer (R30) to the stack pointer (R31) and restoring the original frame pointer from the stack.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format V

Exceptions

None

DIV Divide DIV

Syntax		Instruction	Opcode
div.b	src.b.r, dst.b.rw	Divide Byte	A1
div.h	src.h.r, dst.h.rw	Divide Halfword	A3
div.w	src.w.r, dst.w.rw	Divide Word	A5

Operation

dst ← dst + src

Description

The contents of the destination operand is replaced with the quotient of the source and destination operands. The quotient is computed according to the rules of signed division. Overflow occurs when the negative maximum integer is divided by -1.

The destination will remain unchanged if an integer overflow or Zero Divide exception occurs.

If the Immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	•	•	•

CY Unchanged
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn][Rx]	O	O
disp [Rn/PC][Rx]	O	O
[disp [Rn/PC]][Rx]	O	O
/addr (Rx)	O	O
[/addr][Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

Zero Divide

DIVF Divide Floating DIVF

Syntax		Instruction	Opcode
divf.s	src.s.r, dst.s.rw	Divide Short Real	5C•1B
divf.l	src.l.r, dst.l.rw	Divide Long Real	5E•1B

Operation

dst ← dst + src

Description

The quotient of the source operand and destination operand is stored in the destination operand. Both the integer and floating point condition codes are updated to reflect the result of the operation.

If the destination operand is zero and the source operand a non-zero normalized number, the result is zero with the sign determined by the exclusive OR of the source and destination signs.

Condition Codes

CY	OV	S	Z
•	0	•	•

CY Set if the result is negative and non-zero, otherwise cleared
OV Cleared
S Set if the mantissa sign bit of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
•	•	•	•	•

FIV Set if an invalid operation occurs, otherwise unchanged
FZD Set if division by zero occurs, otherwise unchanged
FOV Set if the result is infinite, otherwise unchanged
FUD Set if the destination result is denormal, otherwise unchanged
FPR Set if a precision error occurs, otherwise unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn][Rx)	O	O
disp [Rn/PC][Rx)	O	O
[disp [Rn/PC]][Rx)	O	O
/addr (Rx)	O	O
[/addr][Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

Instruction Exceptions

Reserved Floating Point Operand
Invalid Floating Point Operation
Floating Point Divide by Zero
Floating Point Overflow
Floating Point Underflow
Floating Point Precision

DIVU Unsigned Divide DIVU

Syntax		Instruction	Opcode
divu.b	src.b.r, dst.b.rw	Divide Unsigned Byte	B1
divu.h	src.h.r, dst.h.rw	Divide Unsigned Halfword	B3
divu.w	src.w.r, dst.w.rw	Divide Unsigned Word	B5

Operation

dst ← dst ÷ src (unsigned)

Description

The contents of the destination operand is replaced with the quotient of the source and destination operands. The quotient is computed according to the rules of unsigned division.

The destination will remain unchanged if a Zero Divide exception occurs.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	*	*

CY	Unchanged
OV	Cleared
S	Set if the result is negative, otherwise cleared
Z	Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-046

Exceptions

Zero Divide

DIVX Divide Extended DIVX

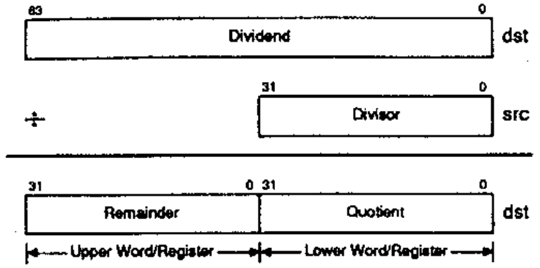
Syntax		Instruction	Opcode
divx	src.w.r, dst.d.rw	Divide Extended	A6

Operation

dst ← dst ÷ src

Description

The doubleword contents of the destination operand is divided by the word contents of the source operand according to the rules of signed division. The resulting 32-bit quotient is stored in the lower word of the destination and the 32-bit remainder is stored in the upper word of the destination.



Overflow occurs when the negative maximum integer is divided by -1. The destination operand does not change when an overflow or a Zero Divide exception occurs.

If the immediate quick addressing mode is specified for the source operand, the data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	*	*	*

CY	Unchanged
OV	Set if integer overflow occurs, otherwise cleared
S	Set if the result is negative, otherwise cleared
Z	Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-046

Exceptions

Zero Divide

DIVUX Unsigned Divide Extended DIVUX

Syntax		Instruction	Opcode
divux	src.w.r, dst.d.rw	Divide Extended Unsigned	B6

Operation

dst ← dst ÷ src (unsigned)

Description

The doubleword contents of the destination operand is divided by the word contents of the source operand according to the rules of unsigned division. The resulting 32-bit quotient is stored in the lower word of the destination and the 32-bit remainder is stored in the upper word of the destination.

The destination operand does not change when an overflow or a Zero Divide exception occurs.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	•	•	•

CY Unchanged
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-044

Exceptions

Zero Divide

EXTBF Extract Bit Field EXTBF

Syntax		Instruction	Opcode
extbfs	bsrc.w.r, blen.b.r, dst.w.w	Extract Sign Extended Bit Field	5D•08
extbfz	bsrc.w.r, blen.b.r, dst.w.w	Extract Zero Extended Bit Field	5D•09
extbfl	bsrc.w.r, blen.b.r, dst.w.w	Extract Left Justified Bit Field	5D•0A

Operation

dst ← bitfield

Description

The designated bit field is extracted using the specified mode and stored in the destination operand.

If the bit field length is zero, zero will be stored in the destination operand.

The sum of the bit offset and the bit field length must not exceed thirty-two, otherwise an Illegal Data Field exception will occur.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIb

Addressing Modes

Bit Address	bsrc	blen	dst	Byte Address
Rn	X	O	O	Rn
@[Rn]	O	–	O	[Rn]
@[Rn+]	O	–	O	[Rn+]
@[–Rn]	O	–	O	[–Rn]
offset@[Rn]	O	–	O	disp[Rn]
offset@[PC]	O	–	O	disp[PC]
@[disp[Rn]]	O	–	O	[disp[Rn]]
@[disp[PC]]	O	–	O	[disp[PC]]
offset@[disp[Rn]]	O	–	O	disp1[disp2[Rn]]
offset@[disp[PC]]	O	–	O	disp1[disp2[PC]]
@/addr	O	–	O	/addr
@[/addr]	O	–	O	[/addr]
Rx@[Rn]	O	–	O	[Rn](Rx)
Rx@[Rn]	O	–	O	disp [Rn](Rx)
Rx@[PC]	O	–	O	disp [PC](Rx)
Rx@[disp[Rn]]	O	–	O	[disp[Rn]](Rx)
Rx@[disp[PC]]	O	–	O	[disp[PC]](Rx)
Rx@/addr	O	–	O	/addr (Rx)
Rx@[/addr]	O	–	O	[/addr](Rx)
Immediate.Quick	X	–	X	Immediate.Quick
Immediate	X	O	X	Immediate

X Illegal Addressing Mode
– Unavailable Addressing Mode

86-063

Exceptions

Illegal Data Field

GETATE Get Area Table Entry GETATE

Syntax		Instruction	Opcode
getate	va.ptr.r, dst.d.w	Get Area Table Entry	05

Operation

dst ← ATE(va)

Description

The contents of specified ATE are transferred to the doubleword destination operand. The virtual address and the section designator register (R28) are used to identify the ATE to be referenced.

If the contents of R28 are 0FFFFFFFFH, the virtual address operand is translated using the current virtual address space. Following the execution of the instruction, the Z flag is updated to reflect the result of the translation operation. The Z flag is cleared if the translation is successful and set if the referenced ATBR/ATLR is invalid.

Otherwise, R28 is assumed to contain a pointer to an area table and a lookup of the specifed ATE is performed. No validity checks are performed on the contents of the ATE.

If the immediate quick addressing mode is specified, the immediate data is zero extended to 32-bit length and used as the virtual address.

This instruction can be executed in either the real or virtual address mode.

Condition Codes

CY	OV	S	Z
–	–	–	•

CY Unchanged
OV Unchanged
S Unchanged
Z Set if the address translation is invalid, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	va	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode 86-078

Exceptions

Privileged Instruction

GETPSW Get PSW GETPSW

Syntax		Instruction	Opcode
getpsw	dst.w.w	Get Program Status Word	F6/7

Operation

dst ← PSW

Description

The contents of the Program Status Word (PSW) are copied to the destination operand.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	dst
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	X
Immediate	X

X Illegal Addressing Mode 86-056

Exceptions

None

GETPTE Get Page Table Entry GETPTE

Syntax		Instruction	Opcode
getpte	va.ptr.r, dst.w.w	Get Page Table Entry	04

Operation

dst ← PTE(va)

Description

The contents of specified PTE are transferred to the word destination operand. The virtual address and the section designator register (R28) are used to identify the PTE to be referenced.

If the contents of R28 are 0FFFFFFFFH, the virtual address operand is translated using the current virtual address space. Following the execution of the instruction, the CY and Z flags are updated to reflect the result of the translation operation. The CY flag will be set if the area is not present (I.e, swapped out to a disk) while the Z flag is set if the referenced address translation fails. If either the Z or CY flags are set, the destination remains unchanged.

Otherwise, R28 is assumed to contain a pointer to an area table and a lookup of the specifed PTE is performed. No validity checks are performed on the contents of the PTE.

If the immediate quick addressing mode is specified, the immediate data is zero extended to 32-bit length and used as the virtual address.

This instruction can be executed in either the real or virtual address mode.

Condition Codes

CY	OV	S	Z
•	–	–	•

CY Set if the area is not present, otherwise cleared
OV Unchanged
S Unchanged
Z Set if the address translation is invalid, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	va	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-078

Exceptions

Privileged Instruction

GETRA Get Real Address GETRA

Syntax		Instruction	Opcode
getra	va.ptr.r, dst.w.w	Get Real Address	03

Operation

dst ← real_address(va)

Description

The physical address of the source virtual address is computed and transferred to the destination operand. The virtual address and the section designator register (R28) are used to identify the area table used to perform the address translation.

If the contents of R28 are 0FFFFFFFFH, the virtual address operand is translated using the current virtual address space. Following the execution of the instruction, the CY and Z flags are updated to reflect the result of the translation operation. The CY flag will be set if the area or page is not present (i.e, swapped out to a disk) while the Z flag is set if the address translation fails (i.e, an invalid area table or the page is I/O mapped). If either the Z or CY flags are set, the destination operand remains unchanged.

Otherwise, R28 is assumed to contain a pointer to an area table and the PTE is located in the specified page table. No validity checks are performed on the contents of the PTE and no data reference is made.

If the immediate quick addressing mode is specified, the immediate data is zero extended to 32-bit length and used as the virtual address.

This instruction can be executed in either the real or the virtual address mode.

Instruction Format

Format I, II

Condition Codes

CY	OV	S	Z
•	–	–	•

CY Set if the area is not present, otherwise cleared
OV Unchanged
S Unchanged
Z Set if the address translation is invalid, otherwise cleared

Addressing Modes

Addressing Mode	va	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-078

Exceptions

Privileged Instruction

HALT Halt HALT

Syntax	Instruction	Opcode
halt	Halt and Wait for Interrupt	00

Operation

halt

Description

The processor halts and waits for an interrupt. Following the execution of the interrupt handler, program execution will continue with the instruction following the HALT instruction.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format V

Exceptions

Privileged Instruction

IN Input IN

Syntax		Instruction	Opcode
in.b	port.b.r, dst.b.w	Input Byte	21
in.h	port.h.r, dst.h.w	Input Halfword	23
in.w	port.w.r, dst.w.w	Input Word	25

Operation

dst ← port

Description

The contents of the specified input port are copied to the destination operand.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	port	dst
Rn	X	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	X	X
Immediate	X	X

X Illegal Addressing Mode 86-081

Exceptions

Privileged Instruction

INC Increment INC

Syntax		Instruction	Opcode
inc.b	dst.b.rw	Increment Byte	D8/9
inc.h	dst.h.rw	Increment Halfword	DA/B
inc.w	dst.w.rw	Increment Word	DC/D

Operation

dst ← dst + 1

Description

The contents of the destination operand are incremented.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

The INC instruction is a shorter encoding for the more general instruction

```
add     #1, dst
```

Condition Codes

CY	OV	S	Z
*	*	*	*

CY Set if a carry is generated, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format III

Addressing Modes

Addressing Mode	dst
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	X
Immediate	X

X Illegal Addressing Mode 86-058

Instruction Exceptions

None

INSBF Insert Bit Field INSBF

Syntax		Instruction	Opcode
insbfr	src.w.r, bdst.w.rw, blen.b.r	Insert Right Justified Bit Field	5D•18
insbfl	src.w.r, bdst.w.rw, blen.b.r	Insert Left Justified Bit Field	5D•19

Operation

bitfield ← src

Description

The source operand is converted to a bit field of specified length and stored in the destination operand.

No transfer will occur if the bit field length is zero.

The sum of the bit offset and the bit field length must not exceed thirty-two, otherwise an Illegal Data Field exception will occur.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the word length before performing the insertion operation.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIc

Addressing Modes

Bit Address	bsrc	blen	src	Byte Address
Rn	X	O	O	Rn
@[Rn]	O	–	O	[Rn]
@[Rn+]	O	–	O	[Rn+]
@[–Rn]	O	–	O	[–Rn]
offset@[Rn]	O	–	O	disp[Rn]
offset@[PC]	O	–	O	disp[PC]
@[disp[Rn]]	O	–	O	[disp[Rn]]
@[disp[PC]]	O	–	O	[disp[PC]]
offset@[disp[Rn]]	O	–	O	disp1[disp2[Rn]]
offset@[disp[PC]]	O	–	O	disp1[disp2[PC]]
@/addr	O	–	O	/addr
@[/addr]	O	–	O	[/addr]
Rx@[Rn]	O	–	O	[Rn](Rx)
Rx@[Rn]	O	–	O	disp [Rn](Rx)
Rx@[PC]	O	–	O	disp [PC](Rx)
Rx@[disp[Rn]]	O	–	O	[disp[Rn]](Rx)
Rx@[disp[PC]]	O	–	O	[disp[PC]](Rx)
Rx@/addr	O	–	O	/addr (Rx)
Rx@[/addr]	O	–	O	[/addr](Rx)
Immediate.Quick	X	–	O	Immediate.Quick
Immediate	X	O	O	Immediate

X Illegal Addressing Mode 86-061
– Unavailable Addressing Mode

Exceptions

Illegal Data Field

JMP Jump JMP

Syntax		Instruction	Opcode
jmp	target.b.ex	Jump	D6/7

Operation

PC ← target

Description

The effective address of the destination is computed and program control is transferred unconditionally to the destination.

The destination operand is treated as byte data for the purpose of computing pointer changes for the autoincrement, autodecrement, or scaled indexed addressing modes.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	target
Rn	X
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	X
Immediate	X

X Illegal Addressing Mode

88-071

Exceptions

None

JSR Jump to Subroutine JSR

Syntax		Instruction	Opcode
jsr	target.b.ex	Jump to Subroutine	E8/9

Operation

temp ← target
[–SP] ← NextPC
PC ← temp

Description

The effective address of the destination is computed and the address of the next instruction is pushed onto the stack. Program control is then transferred to the destination.

The destination operand is treated as byte data for the purpose of computing pointer changes for the autoincrement, autodecrement, or scaled indexed addressing modes.

If the destination operand is addressed using R31 (SP) in conjunction with an autoincrement/autodecrement addressing mode, the result is unpredicable.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	target
Rn	X
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	X
Immediate	X

X Illegal Addressing Mode

88-071

Exceptions

None

LDPR — Load Privileged Register — LDPR

Syntax		Instruction	Opcode
ldpr	src.w.r, regID.w.w	Load Privileged Register	12

Operation

PrivilegedRegister(regID) ← src

Description

The source operand is loaded into the specified privileged register.

ID	Register	Name
0	ISP	Interrupt Stack Pointer
1	L0SP	Level 0 Stack Pointer
2	L1SP	Level 1 Stack Pointer
3	L2SP	Level 2 Stack Pointer
4	L3SP	Level 3 Stack Pointer
5	SBR	System Base Register
7	SYCW	System Control Word
8	TKCW	Task Control Word
15	PSW2	Emulation Mode Program Status Word
16	ATBR0	Area Table Base Register 0
17	ATLR0	Area Table Length Register 0
18	ATBR1	Area Table Base Register 1
19	ATLR1	Area Table Length Register 1
20	ATBR2	Area Table Base Register 2
21	ATLR2	Area Table Length Register 2
22	ATBR3	Area Table Base Register 3
23	ATLR3	Area Table Length Register 3
24	TRMOD	Trap Mode Register
25	ADTR0	Address Trap Register 0
26	ADTR1	Address Trap Register 1
27	ADTRM0	Address Trap Mask Register 0
28	ADTRM1	Address Trap Mask Register 1

86-128

A number of restrictions and precautions relating to the execution of the LDPR instruction are listed below :

- Loading to area table base and length registers clears TLB entries with corresponding section numbers.
- The TLB is cleared if the virtual mode is changed to physical mode in the STCW register.

Instruction execution results are unpredicatable if an invalid register ID is specified.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	regID
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	O
Immediate	O	O

86-088

Exceptions

Privileged Instruction
Illegal Data Field

LDTASK — Load Task — LDTASK

Syntax		Instruction	Opcode
ldtask	list.w.r, TCBptr.w.r	Load Task Context	01

Operation

TaskContext ← [TCB]

Description

The context designated by the TCB pointer is installed as the current task context. The task context consists of:

- General purpose registers (R30-R0)

  The general purpose registers of the new task context is controlled by the list operand. The register list is scanned sequentially from the LSB to the MSB. The bits set in the list operand identify which general purpose registers are restored. Bit 31 of the register list is Reserved for Future Use and must be zero.

RFU	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

86-130

- Area table register pairs

  The area table base and length registers (ATBR0/ATLR0 to ATBR3/ATLR3) are restored as specified by the STCW register if virtual mode is enabled and any TLB entries associated with the updated sections are marked as invalid. In real mode, area table registers are not stored in the task context.

- Stack pointers (L0SP-L3SP)

  The stack pointers enabled for switching in the STCW are restored. If the current context is using the interrupt stack, L0SP will become the new stack pointer.

- Task Control Word (TKCW)

  The TKCW is updated with the Task Control Word for the new context.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	list	TCBptr
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	O
Immediate	O	O

86-083

Exceptions

Privileged Instruction

MOV Move MOV

Syntax		Instruction	Opcode
mov.b	src.b.r, dst.b.w	Move Byte	09
mov.h	src.h.r, dst.h.w	Move Halfword	1B
mov.w	src.w.r, dst.w.w	Move Word	2D
mov.d	src.d.r, dst.d.w	Move Doubleword	3F

Operation

dst ← src

Description

The data designated by the source operand is copied to the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before being transferred to the destination.

On the µPD70616 microprocessor, a Reserved Addressing Mode exception will occur if the immediate or immediate quick addressing mode is specified for a doubleword source operand.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

88-045

Exceptions

None

MOVBS MOve Bit String MOVBS

Syntax		Instruction	Opcode
movbsu	bsrc.b.r, blen.b.r, bdst.b.w	Move Bit String (Upward)	5B•08
movbsd	bsrc.b.r, blen.b.r, bdst.b.w	Move Bit String (Downward)	5B•09

Operation

bdst ← bsrc

Description

The source bit string is copied to the destination bit string. Specifying the direction of the operation allows the correct result to be computed when the two bit strings overlap.

To minimize the interrupt latency time, the MOVBS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

During the execution of the MOVBS instruction, registers R28 and R27 contain pointers to the bytes within the source and destination bit strings to be processed next. Following the execution of the instruction, R28 contains the address of the byte containing the final bit of the source bit string while R27 contains the address of the byte containing the final bit of the destination bit string.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	bsrc	blen	bdst
Rn	X	O	X
@[Rn]	O	–	O
@[Rn+]	O	–	O
@[–Rn]	O	–	O
offset@[Rn/PC]	O	–	O
@[disp [Rn/PC]]	O	–	O
offset@[disp [Rn/PC]]	O	–	O
@/addr	O	–	O
@[/addr]	O	–	O
Rx@[Rn]	O	–	O
Rx@[Rn/PC]	O	–	O
Rx@[disp [Rn/PC]]	O	–	O
Rx@/addr	O	–	O
Rx@[/addr]	O	–	O
Immediate.Quick	X	–	X
Immediate	X	O	X

X Illegal Addressing Mode
– Unavailable Addressing Mode

88-052

Exceptions

None

MOVC — Move Character — MOVC

Syntax		Instruction	Opcode
movcu.b	src.b.r, slen.b.r, dst.b.w, dlen.b.r	Move Byte Character Upward	58•08
movcd.b	src.b.r, slen.b.r, dst.b.w, dlen.b.r	Move Byte Character Downward	58•09
movcu.h	src.h.r, slen.b.r, dst.h.w, dlen.b.r	Move Halfword Character Upward	5A•08
movcd.h	src.h.r, slen.b.r, dst.h.w, dlen.b.r	Move Halfword Character Downward	5A•09

Operation

dst ← src

Description

The source character string is copied to the destination character string. The source and destination length parameters indicate the number of characters to be transferred rather than the number of bytes to be transferred.

Character string transfers are initiated from the head of the strings in the address increment mode and from the tail end of the strings in the address decrement mode. The number of characters copied is the minimum of the source and the destination string lengths.

This instruction is interruptable and resumable with registers R28 and R27 used to maintain the source and destination addresses respectively. Following the execution of the MOVC instruction, these registers contain the address of the next logical character to be transferred.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIa

Addressing Modes

Addressing Mode	src	slen	dst	dlen
Rn	X	O	X	O
[Rn]	O	–	O	–
[Rn+]	O	–	O	–
[–Rn]	O	–	O	–
disp [Rn/PC]	O	–	O	–
[disp [Rn/PC]]	O	–	O	–
disp1 [disp2 [Rn/PC]]	O	–	O	–
/addr	O	–	O	–
[/addr]	O	–	O	–
[Rn](Rx)	O	–	O	–
disp [Rn/PC](Rx)	O	–	O	–
[disp [Rn/PC]](Rx)	O	–	O	–
/addr (Rx)	O	–	O	–
[/addr](Rx)	O	–	O	–
Immediate.Quick	X	–	X	–
Immediate	X	O	X	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

88-046

Exceptions

Illegal Data Field

MOVCF — Move Character with Filler — MOVCF

Syntax		Instruction	Opcode
movcfu.b	src.b.r, slen.b.r, dst.b.w, dlen.b.r	Move Byte Character with Filler Upward	58•0A
movcfd.b	src.b.r, slen.b.r, dst.b.w, dlen.b.r	Move Byte Character with Filler Downward	58•0B
movcfu.h	src.h.r, slen.b.r, dst.h.w, dlen.b.r	Move Halfword Character with Filler Upward	5A•0A
movcfd.h	src.h.r, slen.b.r, dst.h.w, dlen.b.r	Move Halfword Character with Filler Downward	5A•0B

Operation

dst ← src

Description

The source character string is copied to the destination character string. The shorter of the source and destination lengths determines the number of characters to be transferred with any additional positions in the destination string filled using the fill character in R26.

Character string transfers are initiated from the head of the strings in the address increment mode and from the tail end of the strings in the address decrement mode.

This instruction is interruptable and resumable with registers R28 and R27 used to maintain the source and destination addresses respectively. Following the execution of the MOVCF instruction, these registers contain the address of the next logical character to be transferred.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIa

Addressing Modes

Addressing Mode	src	slen	dst	dlen
Rn	X	O	X	O
[Rn]	O	–	O	–
[Rn+]	O	–	O	–
[–Rn]	O	–	O	–
disp [Rn/PC]	O	–	O	–
[disp [Rn/PC]]	O	–	O	–
disp1 [disp2 [Rn/PC]]	O	–	O	–
/addr	O	–	O	–
[/addr]	O	–	O	–
[Rn](Rx)	O	–	O	–
disp [Rn/PC](Rx)	O	–	O	–
[disp [Rn/PC]](Rx)	O	–	O	–
/addr (Rx)	O	–	O	–
[/addr](Rx)	O	–	O	–
Immediate.Quick	X	–	X	–
Immediate	X	O	X	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

88-046

Exceptions

Illegal Data Field

MOVCS Move Character with Stopper MOVCS

Syntax		Instruction	Opcode
movcs.b	src.b.r, slen.b.r, dst.b.w, dlen.b.r	Move Byte Character with Stopper	58•0C
movcs.h	src.h.r, slen.b.r, dst.h.w, dlen.b.r	Move Halfword Character with Stopper	5A•0C

Operation

dst ← src

Description

The source character string is copied to the destination string until the end of the source or destination string is reached or the stop character specified by R26 is detected in the source string. The source and destination length parameters indicate the number of characters to be transferred rather than the number of bytes to be transferred.

This instruction is interruptable and resumable with registers R28 and R27 used to maintain the source and destination addresses respectively. Following the execution of the MOVCS instruction, these registers contain the address of the next logical character to be transferred.

Condition Codes

CY	OV	S	Z
*	–	–	–

CY Cleared if the stop character is found, otherwise set
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIa

Addressing Modes

Addressing Mode	src	slen	dst	dlen
Rn	X	O	X	O
[Rn]	O	–	O	–
[Rn+]	O	–	O	–
[–Rn]	O	–	O	–
disp [Rn/PC]	O	–	O	–
[disp [Rn/PC]]	O	–	O	–
disp1 [disp2 [Rn/PC]]	O	–	O	–
/addr	O	–	O	–
[/addr]	O	–	O	–
[Rn](Rx)	O	–	O	–
disp [Rn/PC](Rx)	O	–	O	–
[disp [Rn/PC]](Rx)	O	–	O	–
/addr (Rx)	O	–	O	–
[/addr](Rx)	O	–	O	–
Immediate.Quick	X	–	X	–
Immediate	X	O	X	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

Illegal Data Field

MOVEA Move Effective Address MOVEA

Syntax		Instruction	Opcode
movea.b	src.b.n, dst.w.w	Move Byte Effective Address	40
movea.h	src.h.n, dst.w.w	Move Halfword Effective Address	42
movea.w	src.w.n, dst.w.w	Move Word Effective Address	44

Operation

dst ← effective_address(src)

Description

The effective address of the source operand is transferred to the destination operand. The source operand is not referenced and remains unchanged.

Separate instructions are provided for byte, halfword and word operands to permit correct computation of effective addresses using the autoincrement, autodecrement and scaled index addressing modes.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	X	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	X	X
Immediate	X	X

X Illegal Addressing Mode 86-047

Exceptions

None

MOVF

Move Floating

Syntax

		Instruction	Opcode
movf.s	src.s.r, dst.s.w	Move Short Real	5C
movf.l	src.l.r, dst.l.w	Move Long Real	5E

Operation

dst ← src

Description

The source operand is copied to the destination operand and the flags updated to reflect the state of the destination.

Condition Codes

CY	OV	S	Z
*	0	*	*

CY Set if the destination is negative and non-zero, otherwise cleared
OV Cleared
S Set if the destination mantissa sign bit is set, otherwise cleared
Z Set if the destination is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
*	–	–	*	–

FIV Set if the destination is a NaN or infinite, otherwise unchanged
FZD Unchanged
FOV Unchanged
FUD Set if the destination is denormal, otherwise unchanged
FPR Unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	Δ
Immediate	Δ

X Illegal Addressing Mode
Δ Reserved Addressing Mode

Exceptions

Reserved Floating Point Operand
Floating Point Underflow

MOVS

Move Sign Extended

Syntax

		Instruction	Opcode
movs.bh	src.b.r, dst.h.w	Move Sign Extended Byte to Halfword	0A
movs.bw	src.b.r, dst.w.w	Move Sign Extended Byte to Word	0C
movs.hw	src.h.r, dst.w.w	Move Sign Extended Halfword to Word	1C

Operation

dst ← sign_extend(src)

Description

The contents of the source operand are sign extended to the destination length and copied to the destination operand.

The source and destination operand lengths differ in this instruction. When specified, the autoincrement, autodecrement and scaled index addressing modes are independently calculated for each operand.

When the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before being sign extended to the destination operand length.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

MOVT — Move Truncated — MOVT

Syntax		Instruction	Opcode
movt.hb	src.h.r, dst.b.w	Move Truncated Halfword to Byte	19
movt.wb	src.w.r, dst.b.w	Move Truncated Word to Byte	29
movt.wh	src.w.r, dst.h.w	Move Truncated Word to Halfword	2B

Operation

dst ← truncate(src)

Description

The contents of the source operand are truncated to the destination operand length and copied to the destination operand. If any of the truncated bits do not match the sign of the result, an integer overflow has occurred and the OV flag is set. In the event of an overflow, the destination operand is replaced with the low order bits of the true result.

The source and destination operand lengths differ in this instruction. When specified, the autoincrement, autodecrement and indexed addressing modes are independently calculated for each operand.

Condition Codes

CY	OV	S	Z
–	*	–	–

CY Unchanged
OV Set if integer overflow occurs, otherwise cleared
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

88-046

Exceptions

None

MOVZ — Move Zero Extended — MOVZ

Syntax		Instruction	Opcode
movz.bh	src.b.r, dst.h.w	Move Zero Extended Byte to Halfword	0B
movz.bw	src.b.r, dst.w.w	Move Zero Extended Byte to Word	0D
movz.hw	src.h.r, dst.w.w	Move Zero Extended Halfword to Word	1D

Operation

dst ← zero_extend(src)

Description

The contents of the source operand are zero extended and copied to the destination operand.

The source and destination operand lengths differ in this instruction. When specified, the autoincrement, autodecrement and scaled index addressing modes are independently calculated for each operand.

When the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the destination operand length.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

88-046

Exceptions

None

MUL Signed Multiply MUL

Syntax		Instruction	Opcode
mul.b	src.b.r, dst.b.rw	Multiply Byte	81
mul.h	src.h.r, dst.h.rw	Multiply Halfword	83
mul.w	src.w.r, dst.w.rw	Multiply Word	85

Operation

dst ← dst • src

Description

The product of the source and destination operands is stored in the destination operand. An overflow occurs when the double length intermediate product does fit within the precision of the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	•	•	•

CY Unchanged
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

MULF Multiply Floating MULF

Syntax		Instruction	Opcode
mulf.s	src.s.r, dst.s.rw	Multiply Short Real	5C•1A
mulf.l	src.l.r, dst.l.rw	Multiply Long Real	5E•1A

Operation

dst ← src • dst

Description

The product of the source operand and destination operand is stored in the destination operand. Both the integer and floating point condition codes are updated to reflect the result of the operation.

If either of the operands is zero and the other operand is either zero or normal, the result is zero with the sign determined by the exclusive OR of the source and destination signs.

Condition Codes

CY	OV	S	Z
•	0	•	•

CY Set if the result is negative and non-zero, otherwise cleared
OV Cleared
S Set if the mantissa sign bit of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
•	–	•	•	•

FIV Set if an invalid operation is attempted, otherwise unchanged
FZD Unchanged
FOV Set if the result is infinite, otherwise unchanged
FUD Set if the result is denormal, otherwise unchanged
FPR Set if a precision error occurs, otherwise unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

Exceptions

Reserved Floating Point Operand
Floating Point Overflow
Floating Point Underflow
Floating Point Precision

MULU Unsigned Multiply MULU

Syntax		Instruction	Opcode
mulu.b	src.b.r, dst.b.rw	Multiply Unsigned Byte	91
mulu.h	src.h.r, dst.h.rw	Multiply Unsigned Halfword	93
mulu.w	src.w.r, dst.w.rw	Multiply Unsigned Word	95

Operation

dst ← dst • src (unsigned)

Description

The product of the unsigned source and destination operands is stored in the destination operand. An overflow occurs when the product cannot fit within the precision of the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	*	*	*

CY Unchanged
OV Set if integer overflow occurs, otherwise cleared
S Set if the MSB of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode 86-044

Exceptions

None

MULX Multiply Extended MULX

Syntax		Instruction	Opcode
mulx	src.w.r, dst.d.rw	Multiply Extended Word	86

Operation

dst ← dst • src

Description

The word designated by the destination operand is multiplied by the word contents of the source operand. The resulting doubleword product is stored in destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	*	*

CY Unchanged
OV Cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode 86-046

Exceptions

None

MULUX Unsigned Multiply Extended MULUX

Syntax		Instruction	Opcode
mulux	src.w.r, dst.d.rw	Multiply Extended Unsigned Word	96

Operation

dst ← dst ∗ src (unsigned)

Description

The unsigned word contents of the destination operand is multiply by the unsigned word contents of the source operand. The resulting doubleword product is stored in the destination.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	•	•

CY Unchanged
OV Cleared
S Set if the MSB of the result is set, otherwise cleared
Z Set if the result is zero; otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

NEG Negate NEG

Syntax		Instruction	Opcode
neg.b	src.b.r, dst.b.w	Negate Byte	39
neg.h	src.h.r, dst.h.w	Negate Halfword	3B
neg.w	src.w.r, dst.w.w	Negate Word	3D

Operation

dst ← 0 – src

Description

The two's complement of the source operand is stored in the destination operand.

Integer overflow will occur if the source operand is the largest negative integer which has no counterpart in the two's complement number system. On overflow, the source operand is copied to the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
•	•	•	•

CY Set if the result is non-zero, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

NEGF Negate Floating NEGF

Syntax		Instruction	Opcode
negf.s	src.s.r, dst.s.w	Negate Short Real	5C•09
negf.l	src.l.r, dst.l.w	Negate Long Real	5E•09

Operation

dst ← −src

Description

The negation of the source operand is stored in the destination operand. Both the integer and floating point condition codes are updated to reflect the result of the operation.

Condition Codes

CY	OV	S	Z
•	0	•	•

CY	Set if the result is negative and non-zero, otherwise cleared
OV	Cleared
S	Set if the mantissa sign bit of the result is set, otherwise cleared
Z	Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
•	–	–	•	–

FIV	Set if an invalid operation is attempted, otherwise unchanged
FZD	Unchanged
FOV	Unchanged
FUD	Set if the result is denormal, otherwise unchanged
FPR	Unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[−Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

Exceptions

Reserved Floating Point Operand
Floating Point Underflow

NOP No Operation NOP

Syntax	Instruction	Opcode
nop	No Operation	CD

Operation

PC ← PC + 1

Description

No action is taken. The NOP instruction can be used to secure a place in the code stream or to create a program delay.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format V

Exceptions

None

NOT

One's Complement

NOT

Syntax		Instruction	Opcode
not.b	src.b.r, dst.b.w	One's Complement Byte	38
not.h	src.h.r, dst.h.w	One's Complement Halfword	3A
not.w	src.w.r, dst.w.w	One's Complement Word	3C

Operation

dst ← ~src

Description

The one's complement of the source operand is stored in the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	•	•

CY Unchanged
OV Cleared
S Set if the MSB of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

88-044

Exceptions

None

NOT1

Complement Bit

NOT1

Syntax		Instruction	Opcode
not1	offset.w.r, base.w.rw	Bit Test and Complement	B7

Operation

CY ← bit(base, offset)
Z ← ~bit(base, offset)
bit(base, offset) ← ~bit(base, offset)

Description

The bit located at the sum of the byte base address and bit offset is tested and then complemented. The CY and Z flags reflect the state of the bit prior to the execution of the instruction.

The location of the designated bit is decided by the base operand. If the register addressing mode is used for the base operand, the designated bit is located within a general purpose register at the specified bit offset. For any other addressing mode, the designated bit is at the specified bit offset from the base address. An Illegal Data Field exception occurs if the bit offset is outside the range 0 to 31.

If the autoincrement or autodecrement addressing mode is specified for the base operand, the base operand is treated as word data and is incremented or decremented by four. When the immediate quick addressing mode is specified, the immediate data is zero extended to word length and used as the bit offset.

Condition Codes

CY	OV	S	Z
•	–	–	•

CY Set if the designated bit is 1, otherwise cleared
OV Unchanged
S Unchanged
Z Set if the designated bit is 0, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	offset	base
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

88-045

Exceptions

Illegal Data Field

NOTBS Negate Bit String NOTBS

Syntax		Instruction	Opcode
notbsu	bsrc.b.r, blen.b.r, bdst.b.w	Negate Bit String (Upward)	5B•0A
notbsd	bsrc.b.r, blen.b.r, bdst.b.w	Negate Bit String (Downward)	5B•0B

Operation

bdst ← ~bsrc

Description

The complement of the source bit string is stored in the destination bit string. Specifying the direction of the operation allows the correct result to be computed when bit strings overlap.

To minimize the interrupt latency time, the ANDBS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

During the execution of the NOTBS instruction, registers R28 and R27 contain pointers to the bytes within the source and destination bit strings to be processed next. Following the execution of the instruction, R28 contains the address of the byte containing the final bit of the source bit string while R27 contains the address of the byte containing the final bit of the destination bit string.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	bsrc	blen	bdst
Rn	X	O	X
@[Rn]	O	–	O
@[Rn+]	O	–	O
@[–Rn]	O	–	O
offset@[Rn/PC]	O	–	O
@[disp [Rn/PC]]	O	–	O
offset@[disp [Rn/PC]]	O	–	O
@/addr	O	–	O
@[/addr]	O	–	O
Rx@[Rn]	O	–	O
Rx@[Rn/PC]	O	–	O
Rx@[disp [Rn/PC]]	O	–	O
Rx@/addr	O	–	O
Rx@[/addr]	O	–	O
Immediate.Quick	X	–	X
Immediate	X	O	X

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

None

OR Logical OR OR

Syntax		Instruction	Opcode
or.b	src.b.r, dst.b.rw	OR Byte	88
or.h	src.h.r, dst.h.rw	OR Halfword	8A
or.w	src.w.r, dst.w.rw	OR Word	8C

Operation

dst ← dst ∨ src

Description

The bit-wise OR of the source and destination operands is stored in the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	*	*

CY Unchanged
OV Cleared
S Set if the MSB of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

ORBS OR Bit String ORBS

Syntax		Instruction	Opcode
orbsu	bsrc.b.r, blen.b.r, bdst.b.rw	OR Bit String (Upward)	5B•14
orbsd	bsrc.b.r, blen.b.r, bdst.b.rw	OR Bit String (Downward)	5B•15

Operation

bdst ← bsrc ∨ bdst

Description

The bit-wise OR of the source and destination bit strings is stored in the destination bit string. Specifying the direction of the operation allows the correct result to be computed when bit strings overlap.

To minimize the interrupt latency time, the ORBS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

During the execution of the ORBS instruction, registers R28 and R27 contain pointers to the bytes within the source and destination bit strings to be processed next. Following the execution of the instruction, R28 contains the address of the byte containing the final bit of the source bit string while R27 contains the address of the byte containing the final bit of the destination bit string.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	bsrc	blen	bdst
Rn	X	O	X
@[Rn]	O	–	O
@[Rn+]	O	–	O
@[–Rn]	O	–	O
offset@[Rn/PC]	O	–	O
@[disp [Rn/PC]]	O	–	O
offset@[disp [Rn/PC]]	O	–	O
@/addr	O	–	O
@[/addr]	O	–	O
Rx@[Rn]	O	–	O
Rx@[Rn/PC]	O	–	O
Rx@[disp [Rn/PC]]	O	–	O
Rx@/addr	O	–	O
Rx@[/addr]	O	–	O
Immediate.Quick	X	–	X
Immediate	X	O	X

X Illegal Addressing Mode
– Unavailable Addressing Mode

Instruction Exceptions

None

ORNBS OR Complemented Bit String ORNBS

Syntax		Instruction	Opcode
ornbsu	bsrc.b.r, blen.b.r, bdst.b.rw	OR Complemented Bit String (Upward)	5B•16
ornbsd	bsrc.b.r, blen.b.r, bdst.b.rw	OR Complemented Bit String (Downward)	5B•17

Operation

bdst ← ~bsrc ∨ bdst

Description

The bit-wise OR of the complemented source bit string and the destination bit string is stored in the destination bit string. Specifying the direction of the operation allows the correct result to be computed when bit strings overlap.

To minimize the interrupt latency time, the ORNBS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

During the execution of the ORNBS instruction, registers R28 and R27 contain pointers to the bytes within the source and destination bit strings to be processed next. Following the execution of the instruction, R28 contains the address of the byte containing the final bit of the source bit string while R27 contains the address of the byte containing the final bit of the destination bit string.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	bsrc	blen	bdst
Rn	X	O	X
@[Rn]	O	–	O
@[Rn+]	O	–	O
@[–Rn]	O	–	O
offset@[Rn/PC]	O	–	O
@[disp [Rn/PC]]	O	–	O
offset@[disp [Rn/PC]]	O	–	O
@/addr	O	–	O
@[/addr]	O	–	O
Rx@[Rn]	O	–	O
Rx@[Rn/PC]	O	–	O
Rx@[disp [Rn/PC]]	O	–	O
Rx@/addr	O	–	O
Rx@[/addr]	O	–	O
Immediate.Quick	X	–	X
Immediate	X	O	X

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

None

OUT Output OUT

Syntax		Instruction	Opcode
out.b	src.b.r, port.b.w	Output Byte	21
out.h	src.h.r, port.h.w	Output Halfword	23
out.w	src.w.r, port.w.w	Output Word	25

Operation

port ← src

Description

The source operand is copied to output port.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	port
Rn	O	X
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode 86-062

Exceptions

Privileged Instruction

POP Pop Word POP

Syntax		Instruction	Opcode
pop	dst.w.w	Pop Word	E6/7

Operation

dst ← [SP+]

Description

The word data located on the top of the stack is copied to the destination operand. The stack pointer (R31) is then incremented by four.

The POP instruction is a shorter encoding of the more general instruction

mov.w [sp+], dst

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	dst
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	X
Immediate	X

X Illegal Addressing Mode [illegible]

Exceptions

None

POPM Pop Multiple POPM

Syntax		Instruction	Opcode
popm	list.w.r	Pop Multiple Registers	E4/5

Operation

registers ← [SP+]

Description

This instruction permits a programmer to pop from 1 to 32 registers from the stack with a single instruction. A register will be restored if and only if its corresponding bit in the register list is set.

PSW	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

86-074

The register list is searched sequentially from the LSB (R0) to the MSB (PSW) with only the designated registers being restored from the stack. If the PSW register is specified, only the lower halfword is modified.

The register list is extended to word length if the immediate quick addressing mode is specified.

Condition Codes

CY	OV	S	Z
R	R	R	R

CY Restored if list:31 is set, otherwise unchanged
OV Restored if list:31 is set, otherwise unchanged
S Restored if list:31 is set, otherwise unchanged
Z Restored if list:31 is set, otherwise unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	list
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	O
Immediate	O

Exceptions

None

PREPARE Prepare Stack Frame PREPARE

Syntax		Instruction	Opcode
prepare	num.w.r	Prepare Stack Frame	DE/F

Operation

tmp ← num
[–SP] ← FP
FP ← SP
SP ← SP – tmp

Description

This instruction is used to dynamically generate a new stack frame upon entry into a procedure. First, the contents of the frame pointer (R30) are saved on the stack and the updated SP is copied into the FP register. Finally, the stack pointer is adjusted by the specified number of bytes to allocate storage for local variables for this instance of the procedure.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format III

Address Modes

Addressing Mode	num
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	O
Immediate	O

86-072

Exceptions

None

PUSH Push Word PUSH

Syntax		Instruction	Opcode
push	src.w.r	Push Word	EE/F

Operation

[−SP] ← src

Description

The stack pointer (R31) is decremented by four and the contents of the source operand are copied onto the stack.

The PUSH instruction is a shorter encoding of the more general instruction

```
mov.w     src, [−sp]
```

Condition Codes

CY	OV	S	Z
−	−	−	−

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	src
Rn	○
[Rn]	○
[Rn+]	○
[−Rn]	○
disp [Rn/PC]	○
[disp [Rn/PC]]	○
disp1 [disp2 [Rn/PC]]	○
/addr	○
[/addr]	○
[Rn](Rx)	○
disp [Rn/PC](Rx)	○
[disp [Rn/PC]](Rx)	○
/addr (Rx)	○
[/addr](Rx)	○
Immediate.Quick	○
Immediate	○

Exceptions

None

PUSHM Push Multiple PUSHM

Syntax		Instruction	Opcode
pushm	list.w.r	Push Multiple Registers	EC/D

Operation

[−SP] ← registers

Description

This instruction permits the programmer to push from 1 to 32 registers on to the stack with a single instruction. A register (PSW, Rn) will be saved if the corresponding bit in the register list is set.

PSW	R30	R29	R28	R27	R26	R25	R24	R23	R22	R21	R20	R19	R18	R17	R16	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

86-074

The register list is searched sequentially from the MSB (PSW) to the LSB (R0) with only the designated registers being pushed onto the stack. The SP (R31) is not saved and following the execution of the instruction points to the last register pushed on the stack.

The register list is extended to zero word length if the immediate quick addressing mode is specified.

Condition Codes

CY	OV	S	Z
−	−	−	−

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	list
Rn	○
[Rn]	○
[Rn+]	○
[−Rn]	○
disp [Rn/PC]	○
[disp [Rn/PC]]	○
disp1 [disp2 [Rn/PC]]	○
/addr	○
[/addr]	○
[Rn](Rx)	○
disp [Rn/PC](Rx)	○
[disp [Rn/PC]](Rx)	○
/addr (Rx)	○
[/addr](Rx)	○
Immediate.Quick	○
Immediate	○

86-087

Exceptions

None

REM Remainder REM

Syntax		Instruction	Opcode
rem.b	src.b.r, dst.b.rw	Remainder Byte	50
rem.h	src.h.r, dst.h.rw	Remainder Halfword	52
rem.w	src.w.r, dst.w.rw	Remainder Word	54

Operation

dst ← dst % src

Description

The integer remainder of the destination operand (dividend) divided by the source operand (divisor) is stored in the destination operand. The sign of the remainder is the same as the sign of the dividend.

The destination operand remains unchanged when a Zero Divide exception occurs.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	•	•

CY Unchanged
OV Cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-044

Exceptions

Zero Divide

REMU Unsigned Remainder REMU

Syntax		Instruction	Opcode
remu.b	src.b.r, dst.b.rw	Unsigned Remainder Byte	51
remu.h	src.h.r, dst.h.rw	Unsigned Remainder Halfword	53
remu.w	src.w.r, dst.w.rw	Unsigned Remainder Word	55

Operation

dst ← dst % src(unsigned)

Description

The remainder of the destination operand (dividend) divided by the source operand (divisor) is stored in the destination operand. All operands are treated as unsigned data, however, the condition code flags are set as if the remainder is a signed value.

The destination operand remains unchanged when a Zero Divide exception occurs.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	•	•

CY Unchanged
OV Cleared
S Set if the MSB of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-048

Exceptions

Zero Divide

RET

Return from Procedure

Syntax		Instruction	Opcode
ret	num.w.h	Return from Procedure	E2/3

Operation

```
tmp1 ← num
tmp2 ← [ SP+ ]
AP ← [ SP+ ]
SP ← SP + tmp1
PC ← tmp2
```

Description

The return address and argument pointer register (R29) are restored from the stack and program control is returned to the calling module. The optional number operand is used to automatically discard any input parameters from the stack.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format III

Addressing Modes

Addressing Mode	num
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	O
Immediate	O

Exceptions

None

RETIS

Return from Interrupt (System)

Syntax		Instruction	Opcode
retis	count.h.r	Return from Interrupt – System	FA/B

Operation

```
PC ← [ SP+ ]
PSW ← [ SP+ ]
SP ← SP + count
```

Description

The PC and PSW are popped from the stack and control resumes at the point of interruption. The count operand allows the interrupt or exception handler to specify the number of bytes to be automatically discarded from the stack.

The RETIS instruction checks for the occurance of the Asynchronous System and Asynchronous Task Traps. If a higher priority exception is detected, processing will not return to the point of interruption but will instead be vectored to the appropiate trap handler.

When the immediate quick addressing mode is specified, the data is zero extended to 16-bit length and used as the count operand.

Condition Codes

CY	OV	S	Z
R	R	R	R

CY	Restored
OV	Restored
S	Restored
Z	Restored

Instruction Format

Format III

Addressing Modes

Addressing Mode	count
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	O
Immediate	O

Exceptions

Privileged Instruction
Illegal Data Field
Asynchronous System Trap
Asynchronous Task Trap

RETIU Return from Interrupt (User) RETIU

Syntax		Instruction	Opcode
retiu	count.h.r	Return from Interrupt (User)	EA/B

Operation

PC ← [SP+]
PSW ← [SP+]
SP ← SP + count

Description

The PC and PSW are popped from the stack and program control is returned to the point of interruption. The count operand allows the interrupt or exception handler to specify the number of bytes to be automatically discarded from the stack.

The RETIU instruction checks for the occurance of Asynchronous System and Asynchronous Task traps. If a higher priority exception is detected, processing will not return to the point of interruption but will instead be vectored to the appropriate trap handler.

An Illegal Data Field exception will occur if the execution level field in the PSW is not 0 and the ISP flag is set or if an attempt is made to return to a more privileged execution level.

Condition Codes

CY	OV	S	Z
R	R	R	R

CY Restored
OV Restored
S Restored
Z Restored

FIV	FZD	FOV	FUD	FPR
R	R	R	R	R

FIV Restored
FZD Restored
FOV Restored
FUD Restored
FPR Restored

Instruction Format

Format III

Addressing Modes

Addressing Mode	count
Rn	O
[Rn]	O
[Rn+]	O
[-Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	O
Immediate	O

Instruction Exceptions

Illegal Data Field
Asynchronous System Trap
Asynchronous Task Trap

ROT Rotate ROT

Syntax		Instruction	Opcode
rot.b	count.b.r, dst.b.rw	Rotate Byte	89
rot.h	count.b.r, dst.h.rw	Rotate Halfword	8B
rot.w	count.b.r, dst.w.rw	Rotate Word	8D

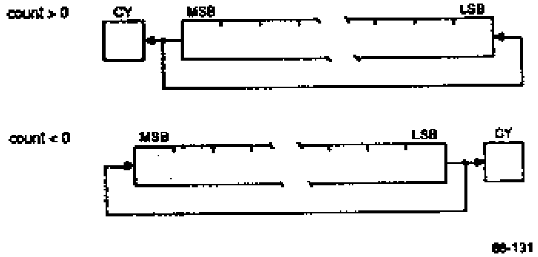
Operation

dst ← rotate(dst, count)

Description

The destination operand is rotated the specified number of bits and stored back in the destination.

The rotate count is specified as signed byte data in a range from -128 to +127. When the rotate count is positive, the destination is rotated left with the contents of the MSB rotating into the LSB. When the rotate count is negative, the destination is rotated right and the contents of the LSB is rotated into the MSB. The destination will remain unchanged if a rotate count of zero is specified but the flags will be updated.



If the immediate quick addressing mode is specified for the count operand, the immediate data is zero extended to byte length before its use as the rotate count.

To minimize the instruction execution time and interrupt latency, the actual rotate count is computed modulo the destination operand size.

Instruction Format

Format I, II

Condition Codes

CY	OV	S	Z
*	0	*	*

CY Set if the last rotated bit was set, cleared if the last rotated bit was zero or the rotate count was zero
OV Cleared
S Set if the MSB of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

Addressing Modes

Addressing Mode	count	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[-Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

ROTC Rotate with Carry ROTC

Syntax		Instruction	Opcode
rotc.b	count.b.r, dst.b.rw	Rotate Byte through Carry	99
rotc.h	count.b.r, dst.h.rw	Rotate Halfword through Carry	9B
rotc.w	count.b.r, dst.w.rw	Rotate Word through Carry	9D

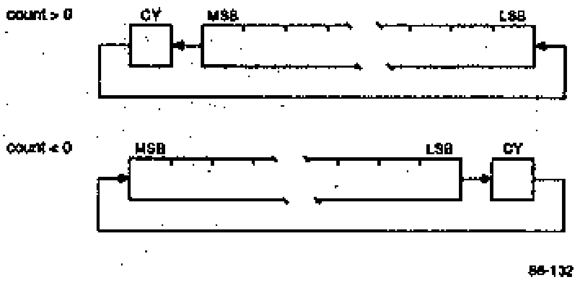
Operation

dst ← rotate_through_carry(dst, count)

Description

The concatentation of the destination operand and CY flag is rotated the specified number of bits and stored back in the destination.

The rotate count is specified as signed byte data in a range from -128 to +127. When the rotate count is positive, the destination is rotated left with the MSB rotating through the CY flag into the LSB. When the rotate count is negative, the destination is rotated right and the contents of the LSB is rotated through the CY flag into the MSB. The destination will remain unchanged if a rotate count of zero is specified but the flags will be updated.



If the immediate quick addressing mode is specified for the count operand, the immediate data is zero extended to byte length before its use as the rotate count.

To minimize the instruction execution time and interrupt latency, the rotate count is computed modulo the destination operand size.

Instruction Format

Format I, II

Condition Codes

CY	OV	S	Z
•	0	•	•

CY	Set if the last rotated bit was set, cleared if the last rotated bit was zero or the rotate count was zero
OV	Cleared
S	Set if the MSB of the result is set, otherwise cleared
Z	Set if the result is zero, otherwise cleared

Addressing Modes

Addressing Mode	count	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[-Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

RSR Return from Subroutine RSR

Syntax	Instruction	Opcode
rsr	Return from Subroutine	CA

Operation

PC ← [SP+]

Description

The return address is popped from the stack and control is returned to the calling module.

The RSR instruction is used to terminate subroutines entered using the JSR and BSR instructions.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format V

Exceptions

None

RVBIT Reverse Bit Order RVBIT

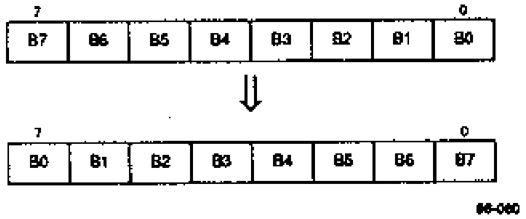
Syntax		Instruction	Opcode
rvbit	src.b.r, dst.b.w	Reverse Bit Order	08

Operation

dst ← bit_reversed(src)

Description

The individual bits of the byte data addressed by the source operand are reversed as follows:



The source operand is unaffected by this instruction.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to byte length before the bit reversal takes place.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

RVBYT Reverse Byte Order RVBYT

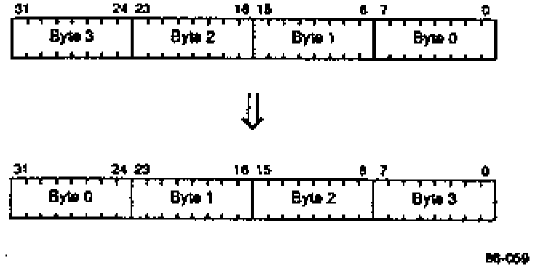
Syntax		Instruction	Opcode
rvbyt	src.w.r, dst.w.w	Reverse Byte Order	2C

Operation

dst ← byte_reversed(src)

Description

The individual bytes of the word data addressed by the source operand are reversed as follows:



The byte order of 16-bit data can be reversed by the ROT instruction.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to word length before the byte reversal takes place.

This instruction is provided to simplify data transfers between machines adopting different integer notations.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

SCH0BS Search Bit String for 0 SCH0BS

Syntax		Instruction	Opcode
sch0bsu	bsrc.b.r, blen.b.r, dst.w.w	Search Bit String for 0 (Upward)	5B•00
sch0bsd	bsrc.b.r, blen.b.r, dst.w.w	Search Bit String for 0 (Downward)	5B•01

Operation

dst ← bit_offset(first 0 bit)

Description

The source bit string is scanned until a zero bit is found or the bit string is exhausted. If found, the bit offset of the detected bit is stored in the destination operand and the Z flag is cleared. Otherwise, the Z flag is set and the bit offset of the next logical bit string after the searched bit string is stored in the destination operand.

To minimize the interrupt latency time, the SCH0BS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

Register R28 is used as a work register during the execution of this instruction, pointing to the current position within the bit string. After the completion of this instruction, R28 will point to the byte containing the detected bit or the byte containing the final bit in the bit string.

Condition Codes

CY	OV	S	Z
–	–	–	•

CY Unchanged
OV Unchanged
S Unchanged
Z Set if a zero bit is not found, otherwise cleared

Instruction Format

Format VIIb

Addressing Modes

Bit Address	bsrc	blen	dst	Byte Address
Rn	X	O	O	Rn
@[Rn]	O	–	O	[Rn]
@[Rn+]	O	–	O	[Rn+]
@[–Rn]	O	–	O	[–Rn]
offset@[Rn]	O	–	O	disp[Rn]
offset@[PC]	O	–	O	disp[PC]
@[disp[Rn]]	O	–	O	[disp[Rn]]
@[disp[PC]]	O	–	O	[disp[PC]]
offset@[disp[Rn]]	O	–	O	disp1[disp2[Rn]]
offset@[disp[PC]]	O	–	O	disp1[disp2[PC]]
@/addr	O	–	O	/addr
@[/addr]	O	–	O	[/addr]
Rx@[Rn]	O	–	O	[Rn](Rx)
Rx@[Rn]	O	–	O	disp [Rn](Rx)
Rx@[PC]	O	–	O	disp [PC](Rx)
Rx@[disp[Rn]]	O	–	O	[disp[Rn]](Rx)
Rx@[disp[PC]]	O	–	O	[disp[PC]](Rx)
Rx@/addr	O	–	O	/addr (Rx)
Rx@[/addr]	O	–	O	[/addr](Rx)
Immediate.Quick	X	–	X	Immediate.Quick
Immediate	X	O	X	Immediate

X Illegal Addressing Mode
– Unavailable Addressing Mode

86-053

Exceptions

None

SCH1BS Search Bit String for 1 SCH1BS

Syntax		Instruction	Opcode
sch1bsu	bsrc.b.r, blen.b.r, dst.w.w	Search Bit String for 1 (Upward)	5B•02
sch1bsd	bsrc.b.r, blen.b.r, dst.w.w	Search Bit String for 1 (Downward)	5B•03

Operation

dst ← bit_offset(first 1 bit)

Description

The source bit string is scanned until a one bit is found or the bit string is exhausted. If found, the bit offset of the detected bit is stored in the destination operand and the. Z flag is cleared. Otherwise, the Z flag is set and the bit offset of the next logical bit string after the searched bit string is stored in the destination operand.

To minimize the interrupt latency time, the SCH1BS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

Register R28 is used as a work register during the execution of this instruction, pointing to the current position within the bit string. After the completion of this instruction, R28 will point to the byte containing the detected bit or the byte containing the final bit in the bit string.

Condition Codes

CY	OV	S	Z
–	–	–	•

CY Unchanged
OV Unchanged
S Unchanged
Z Set if a one bit is not found, otherwise cleared

Instruction Format

Format VIIb

Addressing Modes

Bit Address	bsrc	blen	dst	Byte Address
Rn	X	O	O	Rn
@[Rn]	O	–	O	[Rn]
@[Rn+]	O	–	O	[Rn+]
@[–Rn]	O	–	O	[–Rn]
offset@[Rn]	O	–	O	disp[Rn]
offset@[PC]	O	–	O	disp[PC]
@[disp[Rn]]	O	–	O	[disp[Rn]]
@[disp[PC]]	O	–	O	[disp[PC]]
offset@[disp[Rn]]	O	–	O	disp1[disp2[Rn]]
offset@[disp[PC]]	O	–	O	disp1[disp2[PC]]
@/addr	O	–	O	/addr
@[/addr]	O	–	O	[/addr]
Rx@[Rn]	O	–	O	[Rn](Rx)
Rx@[Rn]	O	–	O	disp [Rn](Rx)
Rx@[PC]	O	–	O	disp [PC](Rx)
Rx@[disp[Rn]]	O	–	O	[disp[Rn]](Rx)
Rx@[disp[PC]]	O	–	O	[disp[PC]](Rx)
Rx@/addr	O	–	O	/addr (Rx)
Rx@[/addr]	O	–	O	[/addr](Rx)
Immediate.Quick	X	–	X	Immediate.Quick
Immediate	X	O	X	Immediate

X Illegal Addressing Mode
– Unavailable Addressing Mode

86-053

Exceptions

None

SCHC Search Character SCHC

Syntax		Instruction	Opcode
schcu.b	src.b.r, slen.b.r, char.b.r	Search Byte Character Upward	58•18
schcd.b	src.b.r, slen.b.r, char.b.r	Search Byte Character Downward	58•19
schcu.h	src.h.r, slen.b.r, char.h.r	Search Halfword Character Upward	5A•18
schcd.h	src.h.r, slen.b.r, char.h.r	Search Halfword Character Downward	5A•19

Operation

R28 ← search character byte address
R27 ← search character offset

Description

The character string is searched for the designated character until either the character is found or all characters in the string have been examined. Character string searches are initiated from the head of the string in the address increment mode and from the tail in the address decrement mode.

This instruction is interruptable and resumable with register R28 used to maintain the character address being scanned. Following the execution of the SCHC instruction, R28 contains the address of the first character meeting the search criteria or the next character after the source string if no matching character was found. Register R27 contains the number of characters (character offset) from the start position to the search end position.

Condition Codes

CY	OV	S	Z
–	–	–	•

CY Unchanged
OV Unchanged
S Unchanged
Z Set if the search character is found, otherwise cleared

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	src	slen	char
Rn	X	O	O
[Rn]	O	–	O
[Rn+]	O	–	O
[–Rn]	O	–	O
disp [Rn/PC]	O	–	O
[disp [Rn/PC]]	O	–	O
disp1 [disp2 [Rn/PC]]	O	–	O
/addr	O	–	O
[/addr]	O	–	O
[Rn](Rx)	O	–	O
disp [Rn/PC](Rx)	O	–	O
[disp [Rn/PC]](Rx)	O	–	O
/addr (Rx)	O	–	O
[/addr](Rx)	O	–	O
Immediate.Quick	X	–	O
Immediate	X	O	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

Illegal Data Field

SCLF Scale Floating SCLF

Syntax		Instruction	Opcode
sclf.s	count.h.r, dst.s.rw	Scale Short Real	5C•10
sclf.l	count.h.r, dst.l.rw	Scale Long Real	5E•10

Operation

dst ← dst • 2^{count}

Description

The destination operand is scaled by the integer count and stored in the destination operand. Both the integer condition codes and the floating point condition codes are updated to reflect the result of the operation.

Condition Codes

CY	OV	S	Z
•	0	•	•

CY Set if the result is negative and non-zero, otherwise cleared
OV Cleared
S Set if the mantissa sign bit of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
•	–	•	•	•

FIV Set if an invalid operation is attempted, otherwise unchanged
FZD Unchanged
FOV Set if the result is infinite, otherwise unchanged
FUD Set if the result is denormal, otherwise unchanged
FPR Set if a precision error occurs, otherwise unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	count	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

Reserved Floating Point Operand
Floating Point Overflow
Floating Point Underflow
Floating Point Precision

SET1 Set Bit SET

Syntax		Instruction	Opco
set1	offset.w.r, base.w.rw	Bit Test and Set	97

Operation

```
CY ← bit( base, offset )
Z ← ~bit( base, offset )
bit( base, offset ) ← 1
```

Description

The bit located at the sum of the byte base address and bit offset is tested and then set. The CY and Z flags reflect the state of the bit prior to the execution of the instruction.

The location of the designated bit is determined by the base operand. If the register addressing mode is used for the base operand, the designated bit is located within a general purpose register at the specified bit offset. For any other permissible addressing mode, the designated bit is at the specified bit offset from the base address. An Illegal Data Field exception occurs if the bit offset is outside the range 0 to 31.

If the autoincrement or autodecrement addressing mode is specified for the base operand, the base operand is treated as word data and is incremented or decremented by four. When the immediate quick addressing mode is specified, the immediate data is zero extended to word length and used as the bit offset.

Condition Codes

CY	OV	S	Z
*	–	–	*

CY Set if the designated bit is 1, otherwise cleared
OV Unchanged
S Unchanged
Z Set if the designated bit is 0, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	offset	bas
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

Illegal Data Field

SETF Set Flag SETF

Syntax		Instruction	Opcode
setf	cond.b.r, dst.b.w	Set Flag Condition	47

Operation

```
if ( condition ) then
    dst ← 01H
else
    dst ← 00H
```

Description

If the specified condition is satisfied by the interger PSW condition codes, the value 01H (true) is stored in the destination. Otherwise, the value 00H (false) is stored in the destintion.

The condition code field is found in the lower four bits of the condition operand. The upper four bits are ignored and have no effect on this instruction.

Encoding	Name	Condition
0000	V	OV = 1
0001	NV	OV = 0
0010	C / L	CY = 1
0011	NC / NL	CY = 0
0100	Z	Z = 1
0101	NZ	Z = 0
0110	NH	(CY ∨ Z) = 1
0111	H	(CY ∨ Z) = 0
1000	S / N	S = 1
1001	NS / P	S = 0
1010	T	Always
1011	F	Never
1100	LT	(S ⊕ OV) = 1
1101	GE	(S ⊕ OV) = 0
1110	LE	((S ⊕ OV) ∨ Z) = 1
1111	GT	((S ⊕ OV) ∨ Z) = 0

Instruction Format

Format I, II

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Addressing Modes

Addressing Mode	cond	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

SHA Arithmetic Shift SHA

Syntax		Instruction	Opcode
sha.b	count.b.r, dst.b.rw	Arithmetic Shift Byte	B9
sha.h	count.b.r, dst.h.rw	Arithmetic Shift Halfword	BB
sha.w	count.b.r, dst.w.rw	Arithmetic Shift Word	BD

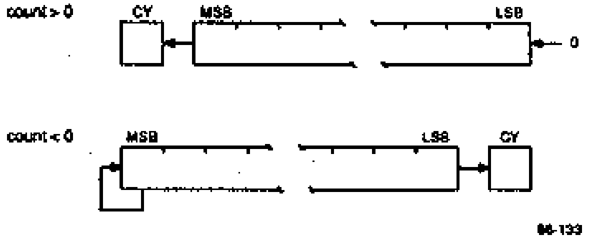
Operation

dst ← arithmetic_shift(dst, count)

Description

The destination operand is arithmetically shifted the specified number of bits and stored back in the destination. Integer overflow occurs if the sign of the result changes at anytime during the execution of this instruction.

The shift count is specified as signed byte data in a range from -128 to +127. When the shift count is positive, the destination is shifted left with a zero bit shifted into the LSB. When the shift count is negative, the destination is shifted right with the MSB being shifted into itself. The destination will remain unchanged if a shift count of zero is specified but the flags will be updated.



If the absolute value of the shift count exceeds the destination operand length, zero (positive shift counts) or data consisting of the sign of the destination (negative shift counts) is stored in the destination.

If the immediate quick addressing mode is specified for the count operand, the immediate data is zero extended to byte length before its use as the shift count.

Instruction Format

Format I, II

Condition Codes

CY	OV	S	Z
*	*	*	*

CY Set if the last shifted bit was set, cleared if the last shifted bit was zero or the shift count was zero
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Addressing Modes

Addressing Mode	count	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[-Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

SHL Logical Shift SHL

Syntax		Instruction	Opcode
shl.b	count.b.r, dst.b.rw	Logical Shift Byte	A9
shl.h	count.b.r, dst.h.rw	Logical Shift Halfword	AB
shl.w	count.b.r, dst.w.rw	Logical Shift Word	AD

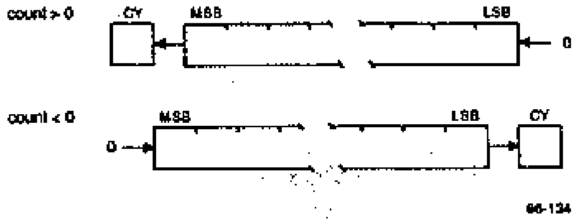
Operation

dst ← logical_shift(dst, count)

Description

The destination operand is logically shifted the specified number of bits and stored back in the destination.

The shift count is specified as signed byte data in a range from -128 to +127. When the shift count is positive, the destination is shifted left with a zero bit shifted into the LSB. When the shift count is negative, the destination is shifted right with a zero bit being shifted into the MSB. The destination will remain unchanged if a shift count of zero is specified but the flags will be updated.



If the absolute value of the shift count exceeds the destination operand length, zero is stored in the destination.

If the immediate quick addressing mode is specified for the count operand, the immediate data is zero extended to byte length before its use as the shift count.

Instruction Format

Format I, II

Condition Codes

CY	OV	S	Z
*	0	*	*

CY Set if the last shifted bit was set, cleared if the last shifted bit was zero or the shift count was zero
OV Cleared
S Set if the MSB of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

Addressing Modes

Addressing Mode	count	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[-Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

SKPC Skip Character SKPC

Syntax		Instruction	Opcode
skpcu.b	src.b.r, slen.b.r, char.b.r	Skip Byte Character Upward	58•1A
skpcd.b	src.b.r, slen.b.r, char.b.r	Skip Byte Character Downward	58•1B
skpcu.h	src.h.r, slen.b.r, char.h.r	Skip Halfword Character Upward	5A•1A
skpcd.h	src.h.r, slen.b.r, char.h.r	Skip Halfword Character Downward	5A•1B

Operation

R28 ← skipped character byte address
R27 ← skipped character offset

Description

The source character string is scanned until a position different from the designated character is reached or the string is exhausted. Character string scanning is initiated from the head of the string in the address increment mode and from the tail in the address decrement mode.

This instruction is interruptable and resumable with register R28 used to maintain the character address being scanned. Following the execution of the SKPC instruction, R28 contains the address of the first character not meeeting the skip criteria or the next character after the source string if the skip criteria was continuously satisfied. Register R27 contains the number of characters (offset) from the start position to the skip end position.

Condition Codes

CY	OV	S	Z
–	–	–	•

CY Unchanged
OV Unchanged
S Unchanged
Z Set if the skip character is found, otherwise cleared

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	src	slen	char
Rn	X	O	O
[Rn]	O	–	O
[Rn+]	O	–	O
[–Rn]	O	–	O
disp [Rn/PC]	O	–	O
[disp [Rn/PC]]	O	–	O
disp1 [disp2 [Rn/PC]]	O	–	O
/addr	O	–	O
[/addr]	O	–	O
[Rn](Rx)	O	–	O
disp [Rn/PC](Rx)	O	–	O
[disp [Rn/PC]](Rx)	O	–	O
/addr (Rx)	O	–	O
[/addr](Rx)	O	–	O
Immediate.Quick	X	–	O
Immediate	X	O	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

86-055

Exceptions

Illegal Data Field

STPR Store Privileged Register STPR

Syntax		Instruction	Opcode
stpr	regID.w.r, dst.w.w	Store Privileged Register	02

Operation

dst ← PrivilegedRegister(regID)

Description

The contents of the specified privileged register are copied to the destination operand.

ID	Register	Name
0	ISP	Interrupt Stack Pointer
1	L0SP	Level 0 Stack Pointer
2	L1SP	Level 1 Stack Pointer
3	L2SP	Level 2 Stack Pointer
4	L3SP	Level 3 Stack Pointer
5	SBR	System Base Register
6	TR	Task Register
7	SYCW	System Control Word
8	TKCW	Task Control Word
9	PIR	Processor ID Register
15	PSW2	Emulation Mode Program Status Word
16	ATBR0	Area Table Base Register 0
17	ATLR0	Area Table Length Register 0
18	ATBR1	Area Table Base Register 1
19	ATLR1	Area Table Length Register 1
20	ATBR2	Area Table Base Register 2
21	ATLR2	Area Table Length Register 2
22	ATBR3	Area Table Base Register 3
23	ATLR3	Area Table Length Register 3
24	TRMOD	Trap Mode Register
25	ADTR0	Address Trap Register 0
26	ADTR1	Address Trap Register 1
27	ADTRM0	Address Trap Mask Register 0
28	ADTRM1	Address Trap Mask Register 1

86-129

An Illegal Data Field exception will occur if the register ID field is not in the range of 0 to 31. Instruction execution results will also be unpredicatable if an undefined register ID is specified.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	regID	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-066

Exceptions

Privileged Instruction
Illegal Data Field

STTASK Store Task STTASK

Syntax		Instruction	Opcode
sttask	list.w.r	Store Task Context	FC/D

Operation

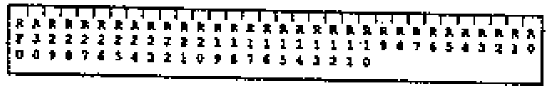
TCB ← TaskContext

Description

The current task context is copied to the Task Control Block (TCB) designated by the Task Register. The task context consists of :

- General Purpose Registers

  The saving of the general purpose registers (R30-R0) is controlled by the list operand. Bits set in the list operand identify which general purpose registers are saved in the TCB. Bit 31 of the register list is Reserved for Future Use and must be zero.



86-130

- Area Table Registers

  In virtual mode, the area table registers (ATBR0/ATLR0-ATBR3/ATLR3) specified by the STCW are saved with the current task context.

- Stack Pointers (L3SP-L0SP)

- Task Control Word (TKCW)

Because no valid context exists between the STTASK instruction and a subsequent LDTASK instruction, the ISP becomes the current stack pointer during the execution of the STTASK instruction.

Instruction Format

Format III

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Addressing Modes

Addressing Mode	list
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	O
Immediate	O

86-097

Exceptions

Privileged Instruction

SUB Subtract SUB

Syntax		Instruction	Opcode
sub.b	src.b.r, dst.b.rw	Subtract Byte	A8
sub.h	src.h.r, dst.h.rw	Subtract Halfword	AA
sub.w	src.w.r, dst.w.rw	Subtract Word	AC

Operation

dst ← dst – src

Description

The source operand is subtracted from the destination operand and the result stored in the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
*	*	*	*

CY	Set if a borrow is generated, otherwise cleared
OV	Set if integer overflow occurs, otherwise cleared
S	Set if the result is negative, otherwise cleared
Z	Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

86-046

Exceptions

None

SUBC Subtract with Carry (Borrow) SUBC

Syntax		Instruction	Opcode
subc.b	src.b.r, dst.b.rw	Subtract Byte with Carry	98
subc.h	src.h.r, dst.h.rw	Subtract Halfword with Carry	9A
subc.w	src.w.r, dst.w.rw	Subtract Word with Carry	9C

Operation

dst ← dst – src – CY

Description

The contents of the source operand and the CY flag are subtracted from the destination operand and the result stored in the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
•	•	•	•

CY Set if a borrow is generated, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

SUBDC Subtract Decimal with Carry SUBDC

Syntax		Instruction	Opcode
subdc	src.b.r, dst.b.rw, pat.b.r	Subtract Decimal with Carry	59•01

Operation

dst ← dst – src – CY using mask pattern

Description

The CY flag and source operand are subtracted from the destination operand with the result stored in the destination operand. The decimal subtraction operation occurs only for the unmasked portion of the data, as determined by the mask pattern.

The CY flag will be set if there is a borrow out of the operation. If the result is not zero or a borrow is generated, the Z flag will be cleared, otherwise it remains unchanged.

Following the subtraction operation, the result is checked to verify that a valid BCD representation exists in the unmasked portion of the result. If either value is not a valid BCD digit (0-9), a Decimal Format exception will occur and the destination will remain unchanged.

Condition Codes

CY	OV	S	Z
•	–	–	•

CY Set if a is generated, otherwise cleared
OV Unchanged
S Unchanged
Z Unchanged if the result is zero, otherwise cleared

Instruction Format

Format VIIc

Addressing Modes

Addressing Mode	src	dst	pat
Rn	O	O	–
[Rn]	O	O	–
[Rn+]	O	O	–
[–Rn]	O	O	–
disp [Rn/PC]	O	O	–
[disp [Rn/PC]]	O	O	–
disp1 [disp2 [Rn/PC]]	O	O	–
/addr	O	O	–
[/addr]	O	O	–
[Rn](Rx)	O	O	–
disp [Rn/PC](Rx)	O	O	–
[disp [Rn/PC]](Rx)	O	O	–
/addr (Rx)	O	O	–
[/addr](Rx)	O	O	–
Immediate.Quick	O	X	–
Immediate	O	X	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

Instruction Exceptions

Decimal Format

SUBF Subtract Floating SUBF

Syntax		Instruction	Opcode
subf.s	src.s.r, dst.s.rw	Subtract Short Real	5C•19
subf.l	src.l.r, dst.l.rw	Subtract Long Real	5E•19

Operation

dst ← src – dst

Description

The difference of the source operand and destination operand is stored in the destination operand. Both the integer and floating point condition codes are updated to reflect the result of the operation.

If the source and destination operands are equal, the sign of the zero result will be determined by the programmed rounding mode.

Condition Codes

CY	OV	S	Z
*	0	*	*

CY Set if the result is negative and non-zero, otherwise cleared
OV Cleared
S Set if the mantissa sign bit of the result is set, otherwise cleared
Z Set if the result is zero, otherwise cleared

FIV	FZD	FOV	FUD	FPR
*	–	*	*	*

FIV Set if an invalid operation is attempted, otherwise unchanged
FZD Unchanged
FOV Set if the result is infinite, otherwise unchanged
FUD Set if the result is denormal, otherwise unchanged
FPR Set if a precision error occurs, otherwise unchanged

Instruction Format

Format II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn][Rx]	O	O
disp [Rn/PC][Rx]	O	O
[disp [Rn/PC]][Rx]	O	O
/addr (Rx)	O	O
[/addr][Rx]	O	O
Immediate.Quick	Δ	X
Immediate	Δ	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

Exceptions

Reserved Floating Point Operand
Floating Point Overflow
Floating Point Underflow
Floating Point Precision

SUBRDC Subtract Decimal Reversed with Carry SUBRDC

Syntax		Instruction	Opcode
subrdc	src.b.r, dst.b.rw, pat.b.r	Subtract Decimal Reversed with Carry	59•02

Operation

dst ← src – dst – CY using mask pattern

Description

The CY flag and destination operand are subtracted from the source operand with the result stored in the destination operand. The decimal subtraction operation occurs only for the unmasked portion of the data, determined by the mask pattern.

The CY flag will be set if there is a borrow out of the operation. If the result is not zero or a borrow is generated, the Z flag will be cleared, otherwise it remains unchanged.

Following the subtraction operation, the result is checked to verify that a valid BCD representation exists in the unmasked portion of the result. If either value is not a valid BCD digit (0-9), a Decimal Format exception will occur and the destination will remain unchanged.

Condition Codes

CY	OV	S	Z
*	–	–	*

CY Set if a borrow is generated, otherwise cleared
OV Unchanged
S Unchanged
Z Unchanged if the result is zero, otherwise cleared

Instruction Format

Format VIIc

Addressing Modes

Addressing Mode	src	dst	pat
Rn	O	O	–
[Rn]	O	O	–
[Rn+]	O	O	–
[–Rn]	O	O	–
disp [Rn/PC]	O	O	–
[disp [Rn/PC]]	O	O	–
disp1 [disp2 [Rn/PC]]	O	O	–
/addr	O	O	–
[/addr]	O	O	–
[Rn][Rx]	O	O	–
disp [Rn/PC][Rx]	O	O	–
[disp [Rn/PC]][Rx]	O	O	–
/addr (Rx)	O	O	–
[/addr][Rx]	O	O	–
Immediate.Quick	O	X	–
Immediate	O	X	O

X Illegal Addressing Mode
– Unavailable Addressing Mode

86-044

Exceptions

Decimal Format

TASI Test and Set TASI

Syntax		Instruction	Opcode
tasi	dst.b.rwi	Test and Set Interlocked	E0/1

Operation

```
lock
flags ← dst – 0FFH
dst ← 0FFH
unlock
```

Description

This instruction is used to synchronize processes or provide mutual exclusion in a multiple processor configuration.

The processor informs the other bus masters in the system that an indivisible operation will take place by asserting the bus lock output. The destination operand is then fetched and compared with 0FFH and the result stored in the condition codes. The contents of the destination operand is then replaced with the value 0FFH and the bus lock output is then negated, allowing other bus masters to again access the shared data.

If the register addressing mode is specified for the destination, the execution of the instruction is meaningless but the operation is carried out.

Condition Codes

CY	OV	S	Z
•	•	•	•

CY Set if a borrow is generated, otherwise cleared
OV Set if integer overflow occurs, otherwise cleared
S Set if the comparison results are negative, otherwise cleared
Z Set if the comparison results are zero, otherwise cleared

Instruction Format

Format III

Addressing Modes

Addressing Mode	dst
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	X
Immediate	X

X Illegal Addressing Mode

Exceptions

None

TB Test and Branch TB

Syntax		Instruction	Opcode
tb	Rn.w.r, disp16	Test and Branch	C7•5

Operation

```
if Rn = 0 then
    PC ← PC + sign_extended( disp16 )
else
    PC ← NextPC
```

Description

The specified general purpose register is tested against zero and if zero, the branch is taken.

The 16-bit displacement field is sign extended to 32 bits and added to the PC to compute the target address. The PC relative addressing mode is implicitedly selected by these instructions. The value of the PC used to compute the target address is the first byte of the test and branch instruction.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VI

Exceptions

None

TEST Test TEST

Syntax		Instruction	Opcode
test.b	src.b.r	Test Byte	F0/1
test.h	src.h.r	Test Halfword	F2/3
test.w	src.w.r	Test Word	F4/5

Operation

flags ← src – 0

Description

Zero is subtracted from the source operand and the result of the operation is reflected in the PSW register.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
0	0	*	*

CY Cleared
OV Cleared
S Set if the result is negative, otherwise cleared
Z Set if the result is zero, otherwise cleared

Instruction Format

Format III

Addressing Modes

Addressing Mode	src
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate,Quick	O
Immediate	O

Exceptions

None

TEST1 Test Bit TEST1

Syntax		Instruction	Opcode
test1	offset.w.r, base.w.r	Bit Test	87

Operation

CY ← bit(base, offset)
Z ← ~bit(base, offset)

Description

The bit located at the sum of the byte base address and bit offset is tested. The CY and Z flags reflect the state of the bit prior to the execution of the instruction.

The location of the designated bit is determined by the base operand. If the register addressing mode is used for the base operand, the designated bit is located within a general purpose register at the specified bit offset. For any other addressing mode, the designated bit is at the specified bit offset from the base address. An Illegal Data Field exception occurs if the bit offset is outside the range 0 to 31.

If the autoincrement or autodecrement addressing mode is specified for the base operand, the base operand is treated as word data and is incremented or decremented by four. When the immediate quick addressing mode is specified, the immediate data is zero extended to word length and used as the bit offset.

Condition Codes

CY	OV	S	Z
*	–	–	*

CY Set if the designated bit is 1, otherwise cleared
OV Unchanged
S Unchanged
Z Set if the designated bit is 0, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	offset	base
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate,Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

Illegal Data Field

TRAP Trap on Condition TRAP

Syntax		Instruction	Opcode
trap	cond&vector.b.r	Trap on Condition	F8/9

Operation

```
if ( condition ) then
    [ -SP ] ← Exception Code
    [ -SP ] ← PSW
    [ -SP ] ← NextPC
    PC ← [ Exception Vector( 48 + vector ) ]
```

Description

If the specified condition is satisfied by the integer condition codes, the specified trap handler is entered.

The upper four bit field of the operand contains the condition code field which indicates under what circumstances the trap will be taken. The lower four bit field contains the vector offset from the software trap base vector.

7 4	3 0
condition code	vector

86-090

Encoding	Name	Condition
0000	V	OV = 1
0001	NV	OV = 0
0010	C / L	CY = 1
0011	NC / NL	CY = 0
0100	Z	Z = 1
0101	NZ	Z = 0
0110	NH	(CY ∨ Z) = 1
0111	H	(CY ∨ Z) = 0
1000	S / N	S = 1
1001	NS / P	S = 0
1010	T	Always
1011	F	Never
1100	LT	(S ⊕ OV) = 1
1101	GE	(S ⊕ OV) = 0
1110	LE	((S ⊕ OV) ∨ Z) = 1
1111	GT	((S ⊕ OV) ∨ Z) = 0

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unaffected
OV	Unaffected
S	Unaffected
Z	Unaffected

Instruction Format

Format III

Addressing Modes

Addressing Mode	cond&vector
Rn	O
[Rn]	O
[Rn+]	O
[–Rn]	O
disp [Rn/PC]	O
[disp [Rn/PC]]	O
disp1 [disp2 [Rn/PC]]	O
/addr	O
[/addr]	O
[Rn](Rx)	O
disp [Rn/PC](Rx)	O
[disp [Rn/PC]](Rx)	O
/addr (Rx)	O
[/addr](Rx)	O
Immediate.Quick	O
Immediate	O

86-079

Exceptions

Software Trap

TRAPFL Trap on Floating Point Exception TRAPFL

Syntax	Instruction	Opcode
trapfl	Trap on Floating Point Exception	CB

Operation

```
if ( TKCW[8:4] ∧ PSW[12:8] ) ≠ 0 then
    Floating Point Operation Exception
```

Description

The bit-wise AND of floating point trap mask field in the TKCW register and the floating point condition codes in the PSW is computed and if the result is non-zero, a floating point operation trap will occur.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

FIV	FZD	FOV	FUD	FPR
–	–	–	–	–

FIV	Unchanged
FZD	Unchanged
FOV	Unchanged
FUD	Unchanged
FPR	Unchanged

Instruction Format

Format V

Exceptions

Floating Point Zero Divide
Invalid Floating Point Operation
Floating Point Overflow
Floating Point Underflow
Floating Point Precision

UPDATE

Update Area Table Entry

UPDATE

Syntax		Instruction	Opcode
update	va.p.r, newATE.d.r	Update Area Table Entry	15

Operation

ATE(va) ← newATE

Description

The contents of specified ATE are replaced with the source doubleword (64-bit) operand. The virtual address and the section designator register (R28) are used to identify the ATE to be referenced.

If the contents of R28 are 0FFFFFFFFH, the virtual address operand is translated using the current virtual address space. Following the execution of the instruction, the Z flag is updated to reflect the result of the translation operation.

Otherwise, R28 is assumed to contain a pointer to an area table and the specifed ATE is located in the area table. No validity checks are performed on the contents of the ATE and if the referenced ATE is cached in the TLB, the entry is invalidated.

If the immediate quick addressing mode is specified for the virtual address operand, the data is zero extended to 32-bit length and used as the virtual address.

This instruction can be executed in either the real or virtual address mode.

Condition Codes

CY	OV	S	Z
–	–	–	*

CY Unchanged
OV Unchanged
S Unchanged
Z Set if the address translation is invalid, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	va	newATE
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	Δ
Immediate	O	Δ

Δ Reserved Addressing Mode

Exceptions

Privileged Instruction

UPDPSW

Update PSW

UPDPSW

Syntax		Instruction	Opcode
updpsw.h	newPSW.w.r, mask.w.r	Update Halfword PSW	4A
updpsw.w	newPSW.w.r, mask.w.r	Update Word PSW	13

Operation

PSW ← (PSW ∧ ~mask) ∨ (newPSW ∧ mask)

Description

The contents of the PSW are updated with the contents of the new PSW image at the positions specified by the mask operand. The UPDPSW.H instruction is restricted to modifying only the condition code fields in the PSW. The UPDPSW.W is a privileged instruction and can also modify the PSW control field.

If the immediate quick addressing mode is specified, the immediate data is zero extended to 32-bit length and used as the new PSW or mask operand.

Condition Codes

Updated according to mask operand

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	newPSW	mask
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	O
Immediate	O	O

Exceptions

Privileged Instruction (updpsw.w)

UPDPTE Update Page Table Entry UPDPTE

Syntax		Instruction	Opcode
updpte	va.p.r, newPTE.w.r	Update Page Table Entry	14

Operation

PTE(va) ← newPTE

Description

The contents of specified PTE are replaced with the source operand. The virtual address and the section designator register (R28) are used to identify the PTE to be referenced.

If the contents of R28 are 0FFFFFFFFH, the virtual address operand is translated using the current virtual address space. Following the execution of the instruction, the CY and Z flags are updated to reflect the result of the translation operation. The CY flag will be set if the area is not present (i.e, swapped out to a disk) while the Z flag is set if the referenced address translation fails. If either the Z or CY flags are set, the destination remains unchanged.

Otherwise, R28 is assumed to contain a pointer to an area table and the specifed PTE is located in the specified page table. No validity checks are performed on the contents of the PTE and if present in the TLB, the entry is invalidated.

When the immediate quick addressing mode is specified, the immediate data is zero extended to 32-bit length and used as the virtual address for new PTE.

This instruction can be executed in either the real or virtual address mode.

Instruction Format

Format I, II

Condition Codes

CY	OV	S	Z
*	–	–	*

CY Set if the area is not present, otherwise cleared
OV Unchanged
S Unchanged
Z Set if the address translation is invalid, otherwise cleared

Addressing Modes

Addressing Mode	va	newPTE
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[–Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	O
Immediate	O	O

Exceptions

Privileged instruction

XCH Exchange XCH

Syntax		Instruction	Opcode
xch.b	dst1.b.rw, dst2.b.rw	Exchange Byte	41
xch.h	dst1.h.rw, dst2.h.rw	Exchange Halfword	43
xch.w	dst1.w.rw, dst2.w.rw	Exchange Word	45

Operation

dst1 ↔ dst2

Description

The contents of the first destination operand is exchanged with the contents of second destination operand.

In the µPD70616 microprocessor, a Reserved Addressing Mode exception will occur if the first destination operand is not a general purpose register.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format I

Addressing Modes

Addressing Mode	dst1	dst2
Rn	O	O
[Rn]	Δ	O
[Rn+]	Δ	O
[–Rn]	Δ	O
disp [Rn/PC]	Δ	O
[disp [Rn/PC]]	Δ	O
disp1 [disp2 [Rn/PC]]	Δ	O
/addr	Δ	O
[/addr]	Δ	O
[Rn](Rx)	Δ	O
disp [Rn/PC](Rx)	Δ	O
[disp [Rn/PC]](Rx)	Δ	O
/addr (Rx)	Δ	O
[/addr](Rx)	Δ	O
Immediate.Quick	X	X
Immediate	X	X

X Illegal Addressing Mode
Δ Reserved Addressing Mode

Exceptions

None

XOR

Exclusive OR

XOR

Syntax		Instruction	Opcode
xor.b	src.b.r, dst.b.rw	Exclusive OR Byte	B0
xor.h	src.h.r, dst.h.rw	Exclusive OR Halfword	B2
xor.w	src.w.r, dst.w.rw	Exclusive OR Word	B4

Operation

dst ← dst ⊕ src

Description

The bit-wise exclusive OR of the source operand and the destination operand is stored in the destination operand.

If the immediate quick addressing mode is specified for the source operand, the immediate data is zero extended to the source operand length before performing the operation.

Condition Codes

CY	OV	S	Z
–	0	•	•

CY	Unchanged
OV	Cleared
S	Set if the result is negative, otherwise cleared
Z	Set if the result is zero, otherwise cleared

Instruction Format

Format I, II

Addressing Modes

Addressing Mode	src	dst
Rn	O	O
[Rn]	O	O
[Rn+]	O	O
[−Rn]	O	O
disp [Rn/PC]	O	O
[disp [Rn/PC]]	O	O
disp1 [disp2 [Rn/PC]]	O	O
/addr	O	O
[/addr]	O	O
[Rn](Rx)	O	O
disp [Rn/PC](Rx)	O	O
[disp [Rn/PC]](Rx)	O	O
/addr (Rx)	O	O
[/addr](Rx)	O	O
Immediate.Quick	O	X
Immediate	O	X

X Illegal Addressing Mode

Exceptions

None

XORBS

XOR Bit String

XORBS

Syntax		Instruction	Opcode
xorbsu	bsrc.b.r, blen.b.r, bdst.b.rw	XOR Bit String (Upward)	5B•18
xorbsd	bsrc.b.r, blen.b.r, bdst.b.rw	XOR Bit String (Downward)	5B•19

Operation

bdst ← bsrc ⊕ bdst

Description

The bit-wise XOR of the source and destination bit strings is stored in the destination bit string. Specifying the direction of the operation allows the correct result to be computed when bit strings overlap.

To minimize the interrupt latency time, the XORBS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

During the execution of the XORBS instruction, registers R28 and R27 contain pointers to the bytes within the source and destination bit strings to be processed next. Following the execution of the instruction, R28 contains the address of the byte containing the final bit of the source bit string while R27 contains the address of the byte containing the final bit of the destination bit string.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY	Unchanged
OV	Unchanged
S	Unchanged
Z	Unchanged

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	bsrc	blen	bdst
Rn	X	O	X
@[Rn]	O	–	O
@[Rn+]	O	–	O
@[−Rn]	O	–	O
offset@[Rn/PC]	O	–	O
@[disp [Rn/PC]]	O	–	O
offset@[disp [Rn/PC]]	O	–	O
@/addr	O	–	O
@[/addr]	O	–	O
Rx@[Rn]	O	–	O
Rx@[Rn/PC]	O	–	O
Rx@[disp [Rn/PC]]	O	–	O
Rx@/addr	O	–	O
Rx@[/addr]	O	–	O
Immediate.Quick	X	–	X
Immediate	X	O	X

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

None

XORNBS XOR Complemented Bit String XORNBS

Syntax		Instruction	Opcode
xornbsu	bsrc.b.r, blen.b.r, bdst.b.rw	XOR Complemented Bit String (Upward)	5B•1A
xornbsd	bsrc.b.r, blen.b.r, bdst.b.rw	XOR Complemented Bit String (Downward)	5B•1B

Operation

bdst ← ~bsrc ⊕ bdst

Description

The bit-wise XOR of the complemented source bit string and the destination bit string is stored in the destination bit string. Specifying the direction of the operation allows the correct result to be computed when bit strings overlap.

To minimize the interrupt latency time, the XORNBS instruction allows the service of interrupts and faults following the completion of a bus cycle. After servicing the interrupt or correction of the fault condition, instruction execution continues from the point of interruption.

During the execution of the XORNBS instruction, registers R28 and R27 contain pointers to the bytes within the source and destination bit strings to be processed next. Following the execution of the instruction, R28 contains the address of the byte containing the final bit of the source bit string while R27 contains the address of the byte containing the final bit of the destination bit string.

Condition Codes

CY	OV	S	Z
–	–	–	–

CY Unchanged
OV Unchanged
S Unchanged
Z Unchanged

Instruction Format

Format VIIb

Addressing Modes

Addressing Mode	bsrc	blen	bdst
Rn	X	O	X
@[Rn]	O	–	O
@[Rn+]	O	–	O
@[-Rn]	O	–	O
offset@[Rn/PC]	O	–	O
@[disp [Rn/PC]]	O	–	O
offset@[disp [Rn/PC]]	O	–	O
@/addr	O	–	O
@[/addr]	O	–	O
Rx@[Rn]	O	–	O
Rx@[Rn/PC]	O	–	O
Rx@[disp [Rn/PC]]	O	–	O
Rx@/addr	O	–	O
Rx@[/addr]	O	–	O
Immediate.Quick	X	–	X
Immediate	X	O	X

X Illegal Addressing Mode
– Unavailable Addressing Mode

Exceptions

None

Section 8
Interrupts and Exceptions

This section describes the interrupt and exception handling capabilities of the µPD70616 microprocessor.

An interrupt is an event which occurs asynchronously to the operation of the µPD70616 while an exception is an event which occurs as a direct result of program execution. When an interrupt or exception is recognized, the program is suspended and control is transferred to an interrupt or exception handler. Processing of interrupts and exceptions is similar but differ slightly in the operation of the interrupt enable flag and the stack pointer used during interrupt or exception processing.

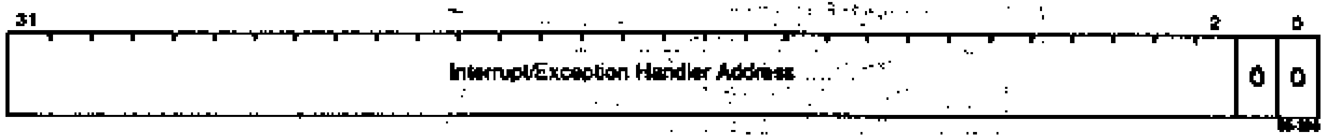
Reset is a special type of exception. When reset occurs, all processor activity is stopped and the processor is initialized to the reset state.

System Base Table

When an interrupt or exception is recognized, the SBT (System Base Table) is used to locate the appropriate vector. The SBT consists of 256 entries each containing a vector to an interrupt or exception handler. An SBT entry consists of a 32-bit virtual address in the virtual address mode. In the physical address mode, an SBT entry consists of a 24-bit physical address with the high order eight bits being ignored by the µPD70616.

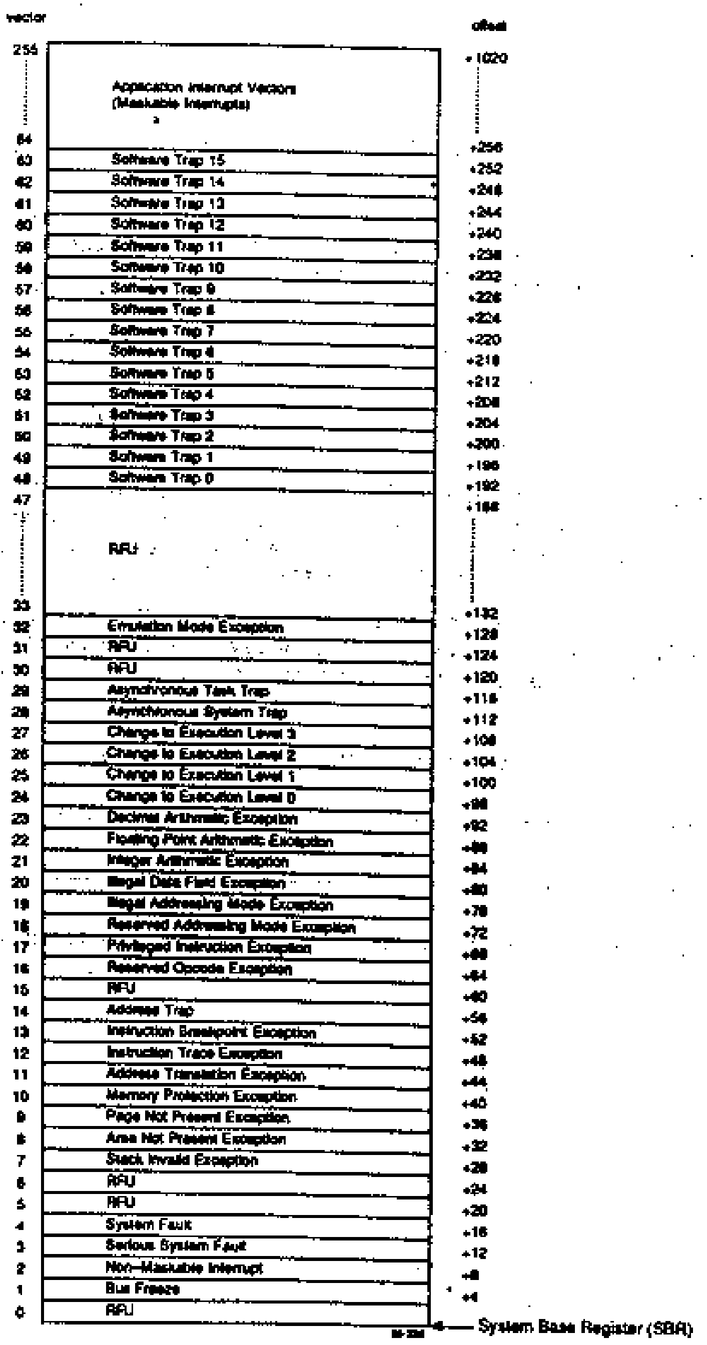
The SBT is located in the memory address space aligned on a page (4KB) boundary by the SBR (System Base Register). The first 64 SBT entries (0–63) are reserved for use by µPD70616 interrupts and exceptions. The remaining 192 entries (64–255) are available in user applications as maskable interrupt vectors.

Figure 8–1. System Base Table Entry



ne starting address of an interrupt/exception handler must be aligned on a word boundary.

Figure 8–2. System Base Table (SBT)



vector		offset
255	Application Interrupt Vectors (Maskable Interrupts)	+1020
64		+256
63	Software Trap 15	+252
62	Software Trap 14	+248
61	Software Trap 13	+244
60	Software Trap 12	+240
59	Software Trap 11	+236
58	Software Trap 10	+232
57	Software Trap 9	+228
56	Software Trap 8	+224
55	Software Trap 7	+220
54	Software Trap 6	+216
53	Software Trap 5	+212
52	Software Trap 4	+208
51	Software Trap 3	+204
50	Software Trap 2	+200
49	Software Trap 1	+196
48	Software Trap 0	+192
47	RFU	+188
33		+132
32	Emulation Mode Exception	+128
31	RFU	+124
30	RFU	+120
29	Asynchronous Task Trap	+116
28	Asynchronous System Trap	+112
27	Change to Execution Level 3	+108
26	Change to Execution Level 2	+104
25	Change to Execution Level 1	+100
24	Change to Execution Level 0	+96
23	Decimal Arithmetic Exception	+92
22	Floating Point Arithmetic Exception	+88
21	Integer Arithmetic Exception	+84
20	Illegal Data Field Exception	+80
19	Illegal Addressing Mode Exception	+76
18	Reserved Addressing Mode Exception	+72
17	Privileged Instruction Exception	+68
16	Reserved Opcode Exception	+64
15	RFU	+60
14	Address Trap	+56
13	Instruction Breakpoint Exception	+52
12	Instruction Trace Exception	+48
11	Address Translation Exception	+44
10	Memory Protection Exception	+40
9	Page Not Present Exception	+36
8	Area Not Present Exception	+32
7	Stack Invalid Exception	+28
6	RFU	+24
5	RFU	+20
4	System Fault	+16
3	Serious System Fault	+12
2	Non–Maskable Interrupt	+8
1	Bus Freeze	+4
0	RFU	

Interrupt and Exception Processing

When an interrupt or exception is recognized, the following actions are performed and control is transferred to the specified interrupt/exception handler.

(i) PSW.EL ← 00 (Note 1)

(ii) PSW.IE flag modification

- interrupt..........PSW.IE ← 0 (maskable interrupts disabled)
- exception.........PSW.IE unchanged (Note 2)

(iii) PSW.TE ← 0
PSW.TP ← 0
PSW.AE ← 0

(iv) PSW.EM ← 0 (native mode)

(v) PSW.ASA ← 1 (Note 3)

(vi) temp ← SBT[vector]

(vii) interrupt/exception information is stored on the stack

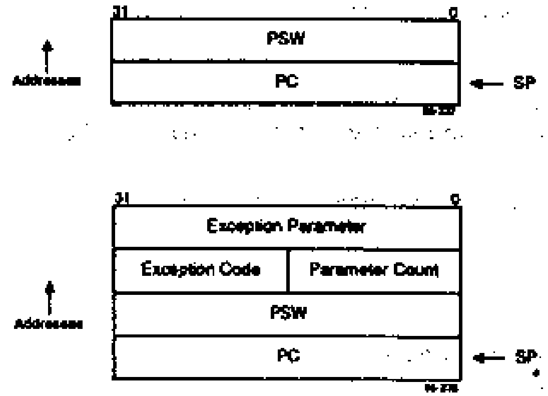
- interrupt..........IS (iInterrupt stack)
- exception.........L0SP (Note 4)

(viii) PC ← temp

The contents of the stack following an interrupt or exception is shown in Figure 8–5.

Figure 8–3. Interrupt/Exception Stack Format

- Interrupt
- Exception



The parameter count indicates the number of bytes of exception information in addition to the PC and PSW in the exception data. It is used by exception handlers to determine the number of bytes to discard from the stack following the processing of the exception.

Note 1 If the exception is caused by the CHLVL instruction or an Asynchronous Task Trap then the specified execution level is set.

Note 2 Bus error and level stack invalid exceptions will disable maskable interrupts.

Note 3 If an ATT (Asynchronous Task Trap) occurs then Asynchronous System Trap is enabled.

Note 4 If the previous stack was the interrupt stack then the IS is continued to be used.

The value of the PC placed on the stack varies depending on the type of exception as follows:

- An exception during the execution of an instruction stacks the PC of the instruction causing the exception (Current PC).
- An exception following the execution of an instruction stacks the PC of the instruction immediately following the instruction which caused the exception (Next PC).

Interrupts

Interrupts are requests for service from external devices and include the fault interrupt, non-maskable and maskable interrupt input signals. Following the acknowledgement of an interrupt, the PC and PSW are saved on the interrupt stack and program control is transferred to the predesignated or supplied vector at execution level 0.

(i) Fault Interrupt

The fault interrupt is used in systems employing functional redundancy monitoring (FRM). If a fault interrupt is detected, the µPD70616 enters the halt state and the address and data buses are placed in the high impedance mode. Actual processing of the fault interrupt is delayed until after the fault interrupt input is negated.

During the processing of a fault interrupt, non-maskable interrupts are disabled.

(ii) Non-Maskable Interrupt (NMI)

Non-maskable interrupts are used to signal the occurrence of catastrophic events such as a loss of power. Non-maskable interrupts cannot be masked by software.

Another non-maskable interrupt will not be acknowledged until the processing of the first NMI completes and the RETIS instruction is executed.

(iii) Maskable Interrupt (MI)

Maskable Interrupt requests are generated by an external interrupt controller. A privileged program can control the recognition of maskable interrupts by the means of the IE (Interrupt Enable) bit in the PSW register.

Following the occurrence of a maskable interrupt, further maskable interrupts will be disabled until the PSW.IE is again set.

Exceptions

Exceptions are divided into the following categories:

- Serious System Exceptions
- System Exceptions
- Stack Invalid Exceptions
- Memory Management Exceptions
- Software Debug Exceptions
- Instruction Exceptions
- Arithmetic Exceptions
- Change Execution Level
- Asynchronous Traps
- Emulation Mode Exceptions
- Software Traps

A list of the possible µPD70616 exceptions and their associated exception stack formats can be found in Table 8–1.

µPD70616 Exception Processing

The complete set of exception conditions are described below. For each possible exception, a short summary of the exception and the control flow for correcting and recovering from the exception is presented.

Serious System Faults

Serious system faults are potentially catastrophic events such as a bus error. When a serious system fault is detected, the exception information is pushed onto the interrupt stack and control is transferred to the serious system fault exception handler.

- Bus Error

  A bus error indicates that a memory or I/O bus cycle has failed and that external hardware is unable to correct the fault.

  In the case of a bus error, the exception address stored on the stack is the physical address that generated the exception. Both maskable and non-maskable interrupts are disabled during the execution of the bus error exception handler until the RETIS instruction is executed.

  When a bus error involves the interrupt stack or a second bus error occurs during the processing of the initial bus error, the situation is deemed unrecoverable and the processor will halt.

System Faults

System faults are exceptions which occur as a result of external events. When a system fault occurs, the exception information is pushed on the interrupt stack and control is transferred to the system fault exception handler.

- Invalid Interrupt

  An invalid interrupt exception occurs when an external interrupt controller supplies a system base table vector in the range of 0 to 63. These interrupt/exception vectors are reserved for system use and attempted use will result in an exception.

Stack Invalid Exceptions

It is necessary to avoid additional exceptions during the processing of an exception. Exceptions involving the level 0 stack and double exceptions are included in this category.

Level 0 Stack Invalid

A level 0 stack invalid exception occurs when a memory management exception is generated when attempting to access the level 0 stack. Because other exceptions normally use the level 0 stack during exception processing, this exception must be handled differently.

When a level 0 stack invalid exception occurs, the stack is switched to the interrupt stack and the level 0 stack invalid exception handler is entered to process the exception.

- Double Exceptions

  When an exception occurs, the stack is switched to the level 0 stack (or other stack in the case of an asynchronous task trap or change execution level exception) and the exception information is saved. A double exception occurs if another exception is encountered while saving the exception information from the first exception on the stack.

When a double exception occurs, the stack is changed to the interrupt stack and the double exception information is stored together with the information from the first exception. The exception handler must then make the stack accessible and copy the first exception information from the interrupt stack to the invalid stack. The RETIS instruction will automatically transfer control to the first exception handler since the return address in exception frame contains the entry point of the first exception handler.

A bus error during the exception information saving will be processed as a bus error and the first exception disregarded.

Memory Management Exceptions

Memory management exceptions are divided into three separate types. Not present exceptions occur when a needed address translation table is not present in memory and must be brought in from secondary storage. Protection exceptions are detected when an instruction attempts to access a location without first having established the proper permissions. Translation exceptions occur when an invalid or out of range translation table entry is referenced during the address translation process.

emory management exception handlers have the option of restarting the faulted instruction or terminating the task if the fault cannot be corrected.

- Area Not Present
- Page Not Present

These exceptions occur if the area table entry or page table entry has a cleared P (Present) bit. The exception handler must read in the area or the page from secondary storage and restart the instruction by executing the RETIS instruction.

- Read Access Violation
- Write Access Violation
- Read/Write Access Violation
- Execute Access Violation
- I/O Access Violation

Access violations occur when the faulted instruction does not have the proper permissions to complete the access. Read, write and execute permissions are checked at both the area and page levels and an access violation will occur if:

- The present execution level is less than the specified ATE access level for the access.

  PSW.EL > { RDL, WRL, EXL }

- The page level permission for the access type is disabled.

An I/O access violation will occur if an access crosses a page boundary and the pages are mapped in different address spaces.

The exception handler processes protection faults by aborting the task or changing the access permissions and restarting the instruction.

- Invalid Section
- Section Length Violation
- Invalid Area
- Area Length Violation
- Invalid Page

Invalid translation faults occur when a section, area or page is marked as not valid. In addition to the validity check, a length check is made on sections and areas to determine if the access exceeds the defined length of the section or area.

The ATBR (Area Table Base Register) and the area table entries and page table entries all contain a V (Valid) bit which the memory management unit uses to determine if the translation is valid. The ATLR (Area Table Length Register) and ATE also contain a length parameter that defines the total size of the section or area. These entries are used to determine if an access beyond the defined region would take place.

The exception handler must either abort the task or allocate additional memory resources and restart the instruction.

Instruction Exceptions

Instruction exceptions involve improper use of an instruction, operand or addressing mode. The exception handler is responsible for analyzing the cause of the fault and either aborting the task or emulating the instruction.

Instruction exception are detected before the instruction executes and the exception information contains the PC of the instruction causing the fault.

- Reserved Opcode

  Reserved opcode exceptions occur when an attempt is made to execute an opcode which is not assigned a valid instruction.

  These opcodes are reserved for future instruction set extensions.

- Reserved Addressing Mode

  Reserved opcode exceptions occur when an attempt is made to use a reserved addressing mode.

  These addressing mode encodings are reserved for future extensions to the addressing modes.

- Illegal Instruction Format

  If an instruction encoding cannot be decoded into one of the seven instruction formats, an illegal instruction format exception will occur.

- Illegal Addressing Mode

  An illegal addressing mode exception occurs when a valid addressing mode is used improperly. An example of an illegal addressing mode exception is an attempt to use an immediate addressing mode as the destination operand in an instruction.

- Privileged Instruction

  A privileged instruction exception will occur if an attempt is made to execute a privileged instruction at an execution level other than level 0.

- Illegal Data Field

  An illegal data field exception occurs when an error is detected in the size of an operand. For example, the bit field data type can range in length from 0 to 32 bits. Should a length greater than 32 bits be specified, an illegal data field exception will occur.

Arithmetic Exceptions

Arithmetic exceptions occur as a result of arithmetic operations. Three different classes of exceptions are supported for each of the integer, floating point and decimal data types.

Integer Exceptions

- Integer Zero Divide

  A zero divide exception occurs when a divisor of zero is used in a divide or remainder instruction. A zero divide exception sets the PSW.OV flag and leaves the destination unmodified.

- Integer Overflow

  Integer overflow occurs when the result of an operation cannot be expressed in the precision of the destination. An integer overflow exception is caused by executing the BRKV instruction with the PSW.OV flag set.

Floating Point Exceptions

- Floating Point Zero Divide

  A floating point zero divide exception occurs when zero is used as divisor, except in the of case of 0 ÷ 0 which is an invalid floating point operation. The PSW.FZD flag will be set if a zero divide takes place.

  Floating point zero divide exceptions are enabled by the TKCW.FZT bit. If this bit is set, then the exception will occur immediately and the destination will remain unchanged. If zero divide exceptions are disabled, an infinite result will be placed in the destination operand and program execution will continue.

- Floating Point Overflow
- Floating Point Underflow

  Floating point overflow and underflow exceptions occur when the destination operand is incapable of representing the magnitude of a floating point result.

  When floating point overflow occurs, the PSW.FOV flag is set. An overflow exception will occur immediately if the TKCW.FOT bit is set or will be delayed and an infinite result will be placed in the destination operand.

  When floating point underflow occurs, the PSW.FUD flag is set. An underflow exception will occur immediately if the TKCW.FUT bit is set or will be delayed and a denormal result will be placed in the destination operand.

When exceptions are enabled, the overflow/underflow condition will result in an immediate exception. If the exception occurs as a result of a arithmetic operation, a corrected exponent is stored in the destination operand. Overflow/underflow exceptions caused by a data type conversion instruction will leave the destination operand unchanged.

	Short Reals	Long Reals
Overflow	–192	–1536
Underflow	+192	+1536

- Floating Point Precision

  A floating point precision exception occurs when the result of an arithmetic or data type conversion operation cannot be exactly expressed in the precision of the destination operand and must be rounded.

  When a precision exception occurs, the PSW.FPR bit is set and the exception will occur if the TKCW.FPT bit is set. The rounded result will stored in the destination operand regardless of the state of the TKCW.FPT flag.

- Invalid Floating Point Operation

  An invalid floating operation exception will occur if one of the following operations is attempted:

  0 ÷ 0
  normal ÷ denormal
  denormal ÷ denormal

  The PSW.FIV flag will be set as a result of an invalid operation. If the exception is enabled, the destination operand remains unchanged. If disabled, a QuietNaN is stored in the destination and execution continues.

- Reserved Floating Point Operand

  A reserved floating point operand exception occurs when a NaN or infinity is used as the operand in an instruction. The destination operand is unchanged as a result of this exception.

Decimal Exceptions

- Decimal Format

  A decimal format exception occurs when the result of a decimal arithmetic operation or data type conversion is not a valid BCD representation.

Software Debug Exceptions

- Instruction Trace

  When enabled, instruction trace exceptions occur following the execution of each instruction. Because of the possibility of multiple exceptions and the subsequent restart of an instruction, the TE (Trace Enable) flag is supplemented by the TP (Trace Pending) flag.

  Prior to entering the exception handler, both the PSW.TE and PSW.TP fields are cleared.

- Instruction Breakpoint

  An instruction breakpoint exception occurs when the BRK instruction is executed.

The PC image in the exception information contains the address of the instruction breakpoint. This allows the exception handler to restart the instruction following the removal of the breakpoint.

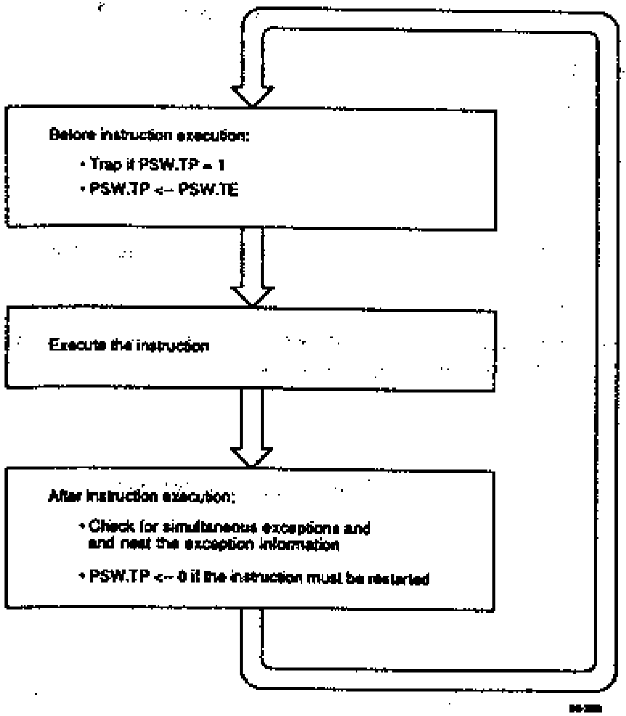
- Address Trap

  An address trap occurs when an access occurs that meets the trap conditions in either of the two address trap registers and address traps are enabled in the PSW. Address traps are controlled by the following registers:

  - TRMOD——access types
  - ADTR———trap base address
  - ADTMR——trap address range

The AE (Address Trap Enable) flag in the PSW is cleared before entering any interrupt or exception handler. The exception handler can determine which address trap occurred by examination of the address trap exception code.

Figure 8-4 Exception Detection Sequence


Before instruction execution:
• Trap if PSW.TP = 1
• PSW.TP <-- PSW.TE
Execute the instruction
After instruction execution:
• Check for simultaneous exceptions and and neat the exception information
• PSW.TP <-- 0 if the instruction must be restarted

Change Execution Level Exceptions

The change execution level exceptions are provided to allow unprivileged tasks access to system functions and devices in a controlled manner.

- Change to Execution Level 0
- Change to Execution Level 1
- Change to Execution Level 2
- Change to Execution Level 3

  The stack is switched to the execution level specified in the instruction and the exception information is saved. The second operand of the instruction is zero extended to word length and used as the exception parameter.

Asynchronous Traps

Asynchronous traps are used to notify the system or a task of the occurrence of an important event. Asynchronous traps are important in the design of operating systems because the trap can be immediate or may be delayed until a ecified execution level is reached. Two types of asynchronous traps are supported. The AST (Asynchronous System Trap) is used to inform the operating system of an event while the ATT (Asynchronous Task Trap) is used to inform a task of an event.

Asynchronous traps are detected by the RETIS/RETIU instructions. During the execution of these instructions, the SYCW.AST and TKCW.ATT fields are compared with the execution level of the destination. If the new execution level is greater or equal to the AST/ATT level the asynchronous trap will occur.

Asynchronous system and task traps operate similarty but differ slightly in the execution level of the trap handler.

Trap	Execution level after the trap
AST	level 0
ATT	level determined by the ATT

<u>AST/ATT Level</u>

0	0	0	................level 0 AST/ATT
0	0	1	................level 1 AST/ATT
0	1	0	................level 2 AST/ATT
0	1	1	................level 3 AST/ATT
1	0	0	................AST/ATT disabled
1	0	1	................AST/ATT disabled
1	1	0	................AST/ATT disabled
1	1	1	................AST/ATT disabled

- Asynchronous System Trap

  The AST (Asynchronous System Trap) is a means of informing the operating system of events. When an interrupt or exception occurs, it is desirable to minimize the time that interrupts are disabled. The exception handler need only set the AST level to the user level and exit. When control is return to the user level the AST will occur and the operating system entered. This technique simplifies the software interface between interrupt/exception handlers and the operating system.

  Following the detection of an AST, the execution level is changed to level 0 and control is transferred to the AST handler.

An asynchronous system trap will be disregarded while the PSW.ASA field indicates an earlier AST is being serviced.

- Asynchronous Task Trap

  The ATT (Asynchronous Task Trap) is a means for entering user trap handlers. Because user trap handlers cannot be allowed to execute at level 0, the ATT is a means to permit task specific trap handlers to execute at non-privileged execution levels. Applications of asynchronous task traps include activation of task exception handlers or task termination processing.

  Following the detection of an ATT, the execution level is changed to the level specified by the TKCW.ATT field and control is transferred to the ATT handler.

  During context switching, the ATT is stored as part of the task context in the TCB by the STTASK instruction and loaded from the TCB by the LDTASK instruction.

  Asynchronous task traps are disregarded when:

  - an AST and ATT occur simultaneously,
  - the PSW.ASA field indicates an AST is being serviced,
  - the PSW.ATA field indicates an ATT is being serviced.

Emulation Mode Exceptions

Emulation mode exceptions are exceptions which occur during the execution of emulation mode programs. Following the detection of an exception, the emulation mode PC is pushed on the stack and native mode is enabled to process the exception. The exception handler must examine the exception parameter and determine the correct method of emulation for each instance.

- Emulation Mode Privileged Instruction

  The emulation mode system instructions listed below have been made privileged to allow operation in a protected environment:

  IN*, OUT*, INM, OUTM, HALT
  BRK, RETI, POP PSW*, EI, DI

  The PC stored with the exception information is the address of the byte immediately following the first byte of the instruction causing the exception.

- Emulation Mode Reserved Opcode

  This exception occurs if a reserved but unimplemented opcode is attempted to be executed. Included in the list of reserved opcodes is the BRKEM (Break for Emulation) instruction.

  The PC stored with the exception information is the address of the byte immediately following the first byte of the instruction causing the exception.

- Emulation Mode Zero Divide

  A zero divide exception has occurred in emulation mode.

  The PC stored with the exception information is the address of the next instruction.

*Conditionally trapped, see Section 10 for details.

- Emulation Mode Single Step Trap

  This exception occurs when an instruction executes and modifies the emulation mode PSW2.BRK flag.

  The PSW2 BRK flag has no effect on single step and the native mode PSW.TE flag must be used to single step emulation mode programs.

- Emulation Mode Overflow

  This exception occurs when the emulation mode BRKV instruction is executed with the PSW2.OV flag set.

- Emulation Mode Index

  This exception occurs when the emulation mode CHKIND (Check Index) instruction is executed and the array index is found to be out of range.

- Emulation Mode Coprocessor Not Present

  This exception occurs if an FPO1 or FPO2 instruction is executed and an external coprocessor is not connected.

Software Traps

Software traps are an implementation dependent method of implementing user traps.

- Software Trap 0 – 15

  When the condition field in a TRAP instruction and the PSW is satisfied, the specified software trap will occur.

Interrupt/Exception Stack Formats

Figure 8-5. µPD70616 Interrupt/Exceptions
(the @ symbol indicates a pointer value)

Bus Freeze Interrupt — Abort — Interrupts Disabled

#1 Bus Freeze Interrupt

+4	PSW
0	PC (Current PC)

Interrupt — Continue — Interrupts Disabled

#2 Non-Maskable Interrupt
#64–255 Maskable Interrupts

+4	PSW
0	PC (Next PC)

Serious System Faults — Abort — Interrupts Disabled

#3 Bus Fault
(Note : The exception address is a physical address)

+12	Exception Address	
+8	Exception Code	8
+4	PSW	
0	PC	

System Exceptions — Abort / Ignore — Interrupts Disabled

#4 System Fault
Invalid Interrupt

+8	Exception Code	4
+4	PSW	
0	PC	

Level 0 Stack Invalid — Abort / Retry — Interrupts Disabled

#7 Level 0 Stack Invalid
Single Exception

+12	Exception Address	
+8	Exception Code	8
+4	PSW	
0	PC (Current PC)	

Double Exception

+16	First Exception Information	
+12	Exception Address	
+8	Exception Code	8
+4	PSW	
0	@ First Exception Handler	

Memory Management Exceptions — Retry / Abort —

#8 Area Not Present Exception
#9 Page Not Present Exception
#10 Memory Management Exceptions
- I/O Access Violation
- Read Access Violation
- Write Access Violation

#11 Address Translation Exceptions
- Invalid Section
- Section Length Violation
- Invalid Area
- Area Length Violation
- Invalid Page

+12	Exception Address	
+8	Exception Code	8
+4	PSW	
0	PC (Current PC)	

Software Debug Exceptions — Continue —

#12 Instruction Trace Exception

+8	Exception Code	4
+4	PSW	
0	PC (Next PC)	

#13 Instruction Breakpoint Exception

+8	Exception Code	4
+4	PSW	
0	PC (Current PC)	

#14 Address Trap

+12	PC (Current PC)	
+8	Exception Code	8
+4	PSW	
0	PC (Next PC)	

Instruction Exceptions — Abort / Emulate —

#16 Reserved Opcode
#17 Privileged Instruction
#18 Reserved Addressing Mode
#19 Illegal Addressing Mode
Illegal Instruction Format

#20 Illegal Data Field

+8	Exception Code	4
+4	PSW	
0	PC (Current PC)	

Arithmetic Exceptions — Abort / Continue —

#21 Integer Exceptions
Zero Divide
Overflow

#22 Floating Point Exceptions
Zero Divide
Overflow
Underflow
Precision
Invalid Floating Point Operation
Reserved Floating Point Operand

#23 Decimal Exceptions
Decimal Format

+12	PC (Current PC)	
+8	Exception Code	8
+4	PSW	
0	PC (Next PC)	

Change Execution Level Exceptions — Continue —

#24 Change to Execution Level 0
#25 Change to Execution Level 1
#26 Change to Execution Level 2
#27 Change to Execution Level 3

+12	Parameter	
+8	Exception Code	8
+4	PSW	
0	PC (Next PC)	

Asynchronous Traps — Continue —

#28 Asynchronous System Trap
#29 Asynchronous Task Trap

+8	Exception Code	4
+4	PSW	
0	PC (Next PC)	

Emulation Mode Exceptions — Abort / Continue —

#32 Emulation Mode Exceptions
Privileged Instruction
Reserved Opcode
Zero Divide
Single Step Trap
Overflow
Array Bounds
CP Not Present

+12	PC (Current PC)	
+8	Exception Code	8
+4	PSW	
0	PC (Next PC)	

Software Traps	— Continue —

#48–63 Software Traps

+8	Exception Code	4
+4	PSW	
0	PC (Next PC)	

Exception Codes

Table 8–1. µPD70616 Exception Codes

Code	Serious System Exceptions	
0301	string data write bus error	
0303	fixed length data write bus error	
0305	translation table write bus error	
0309	string I/O write bus error	
030B	fixed length I/O write bus error	
0311	string data read bus error	
0313	fixed length data read bus error	
0314	system base table read bus error	
0315	translation table read bus error	
0317	instruction fetch bus error	
0319	string I/O read bus error	
031B	fixed length I/O read bus error	
031E	interrupt vector read bus error	
	System Exceptions	
0400	illegal interrupt	
	Stack Invalid Exceptions	
0700	area not present	level 0 stack invalid
0701	page not present	
0702	I/O access violation	
0703	read access violation	
0704	write access violation	
0705	read/write access violation	
0707	invalid section	
0708	section length violation	
0709	invalid area	
070A	area length violation	
070B	invalid page	
0780	area not present	double exception
0781	page not present	
0782	I/O access violation	
0783	read access violation	
0784	write access violation	
0785	read/write access violation	
0787	invalid section	
0788	section length violation	
0789	invalid area	
078A	area length violation	
078B	invalid page	
	Memory Management Exceptions	
0800	area not present	
0901	page not present	
0A02	I/O access violation	
0A03	read access violation	
0A04	write access violation	
0A05	read/write access violation	
0A06	execute access violation	
0B07	invalid section	
0B08	section length violation	
0B09	invalid area	
0B0A	area length violation	
0B0B	invalid page	
Code	**Software Debug Exceptions**	
0C00	instruction trace	
0D00	instruction breakpoint	
0E01	address trap 0	
0E02	address trap 1	
0E03	address traps 0 and 1	
	Instruction Exceptions	
1000	reserved instruction	
1100	privileged instruction	
1200	reserved address mode	
1300	illegal addressing mode	
1301	illegal instruction format	
1400	illegal data field	
	Arithmetic Exceptions	
1500	integer zero divide	
1501	integer overflow	
1601	floating point precision	these exception codes can combine in the case of simultaneous exceptions
1602	floating point underflow	
1604	floating point overflow	
1608	floating point zero divide	
1610	invalid floating point operation	
1680	reserved floating point operand	
1780	decimal format exception	
	Change Execution Level Exceptions	
1800	change to execution level 0	
1900	change to execution level 1	
1A00	change to execution level 2	
1B00	change to execution level 3	
	Asynchronous Traps	
1C00	asynchronous system trap	
1D00	asynchronous task trap	
	Emulation Mode Exceptions	
2000	emulation mode privileged instruction	
2001	emulation mode reserved instruction	
2002	emulation mode zero divide	
2003	emulation mode single step trap	
2004	emulation mode overflow	
2005	emulation mode index	
2006	emulation mode coprocessor not present	
	Software Traps	
3000	software trap 0	
3100	software trap 1	
3200	software trap 2	
⋮	⋮	
3F00	software trap 15	

Reset

The µPD70616 is reset when the RESET input pin is asserted. When reset occurs, internal registers and bus interface are initialized and enter the reset state. Following the negation of RESET, the µPD70616 will begin program execution at address 0FFFFFFF0H.

Following reset, the processor is in the physical address mode using the native mode instruction set. The program execution level is set to level 0 using the interrupt stack and maskable interrupts are disabled. The internal state following reset is shown in Figure 8–6.

Figure 8–6. µPD70616 Reset State

Register	Contents
PC	FFFFFFF0H
PSW	10000000H
R0 – R31	Undefined
L0SP – L3SP, ISP	Undefined
SYCW	00000070H
TKCW	0000E000H
SBR	00000000H
TR	Undefined
ATBR0 – ATBR3	Invalid
ATLR0 – ATLR3	Undefined
ADTR0/1, ADTMR0/1, TRMOD	Undefined
PSW2	0000F002H

Interrupt/Exception Nesting

Multiple interrupts and exceptions can occur simultaneously. In the event of multiple exceptions or interrupts, the µPD70616 assigns each a priority and nests the exception information on the stack according to the priorities of each interrupt or exception.

In general, when nesting multiple exceptions, the exception stack frames are pushed onto the stack in order of increasing priority while the processing of the exceptions will occur in order of decreasing priority. When stacking multiple exception information, the first PC pushed on the stack is a pointer to the current or next instruction. Subsequent PC images are pointers to the exception handlers to be executed in order of their priority.

There are some cases which are handled slightly differently than described above:

- If a reset, bus freeze or serious system exception occurs, all other interrupts and exceptions are ignored. Reset is the highest priority event and takes precedence over all other interrupts and exceptions.
- If a maskable and non-maskable interrupt occur simultaneously, only the non-maskable interrupt will be recognized. The servicing of the maskable interrupt condition will delay until the completion of the non-maskable interrupt handler.
- An address trap will be ignored if it occurs in conjunction with an exception that will restart instruction execution.
- An instruction trace exception will be ignored if it occurs in conjunction with an exception that will restart instruction execution. The instruction trace exception frame is not stored and the PSW field in the other exception frame will have the TP (Trace Pending) bit cleared, preventing an instruction trace exception from occurring when the instruction is restarted.
- Memory management exceptions are not nested with other exceptions. If a memory management exception or address trap occurs simultaneously with a maskable interrupt, it will be ignored and the instruction later restarted.

Table 8–2 is a complete list of the combinations of simultaneous interrupts and exceptions. In this table, interrupts and exceptions are listed across the top and down in order of increasing priority. The action taken in the case of two or more simultaneous interrupts or exceptions can be determined by finding the first exception in the top row and reading down until the column intersects the desired interrupt/exception.

For example, in the case of the instruction trace exception, an arithmetic exception, address trap, maskable and non-maskable interrupt will all be recognized and serviced in order of priority. If an instruction/memory management xception or a serious system fault, bus freeze or reset interrupt occur, the instruction trace exception will be ignored.

Table 8–2. Multiple Interrupt/Exception Processing

Increasing Priority →

Increasing Priority ↓

Interrupt/Exception → Possible Simultaneous Interrupts/Exceptions	ARITH	INSTR	MME	AD-TR	MI	NMI	SERI	BFREZ	RESET
SINGLE	O	●	●	O	O	O	●	●	●
ARITH		–	–	O	O	O	●	●	●
INSTR			–	*	O	O	●	●	●
MME				*	O	O	O	●	●
AD-TR					O	O	●	●	●
MI						●	●	●	●
NMI							●	●	●
SERI								●	●
BFREZ									●

TRACE....Instruction Trace Exception

ARITH.....Arithmetic Exception, Change Level Exception, Asynchronous Trap, Emulation Mode Exception, Software Trap

INSTR.....Instruction Exception, Instruction Breakpoint Exception

MME.......Memory Management Exception

AD-TR.....Address Trap

MI..........Maskable Interrupt, Illegal Interrupt

NMI.........Non-Maskable Interrupt

SERI.......Serious System Exception

BFREZ....Bus Freeze Interrupt

RESET....Reset

O............Both exceptions are recognized and are serviced in order of priority.

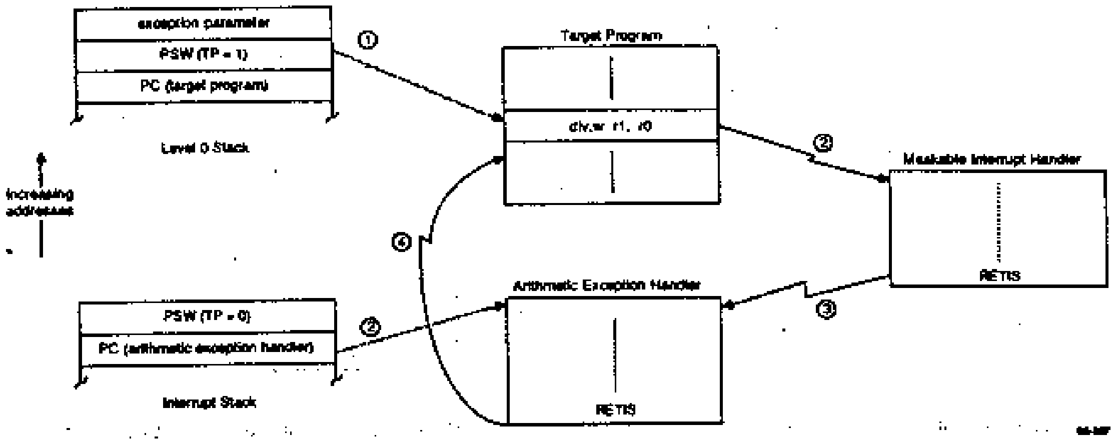
●............The higher priority interrupt/exception is serviced and the lower priority exception is ignored.

–............The occurrence of both exceptions is not possible.

*............The lower priority exception is serviced and the higher priority exception is ignored.

Specific examples of interrupt/exception nesting are shown below:

Figure 8–7 Simultaneous Instruction Trace, Arithmetic and Non-Maskable Interrupt Exception Processing



This example shows the operation of the priority selection logic in servicing simultaneous interrupts and exceptions.

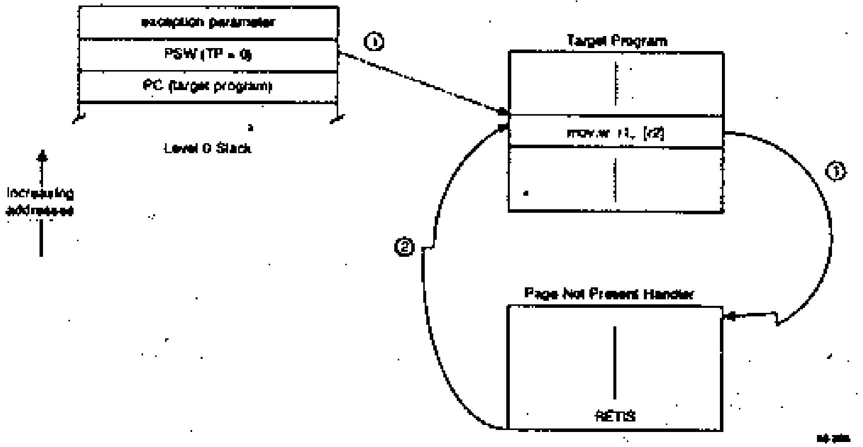
1. The first action taken is to push the arithmetic exception frame on the level 0 stack. Because an instruction trace exception also occurred, the PSW image will have the TP bit set. The TE and TP fields in the PSW are then cleared during exception processing.

2. Next the stack is switched to the interrupt stack for the storage of the interrupt frame. The PC image in the interrupt frame is the entry point of the arithmetic exception handler which is scheduled to be serviced next following the completion of the maskable interrupt handler. The PSW.IE field is cleared to disable further maskable interrupts.

3. When the maskable interrupt handler terminates, the RETIS instruction pops the interrupt frame from the interrupt stack and transfers control to the arithmetic exception handler.

4. The arithmetic exception handler is now using the execution level 0 stack. When the exception condition has been corrected, the arithmetic exception handler executes an RETIS instruction. The instruction trace handler is then entered since the PSW.TP field is set.

Figure 8–8 Simultaneous Instruction Trace and Page Not Present Exception Processing

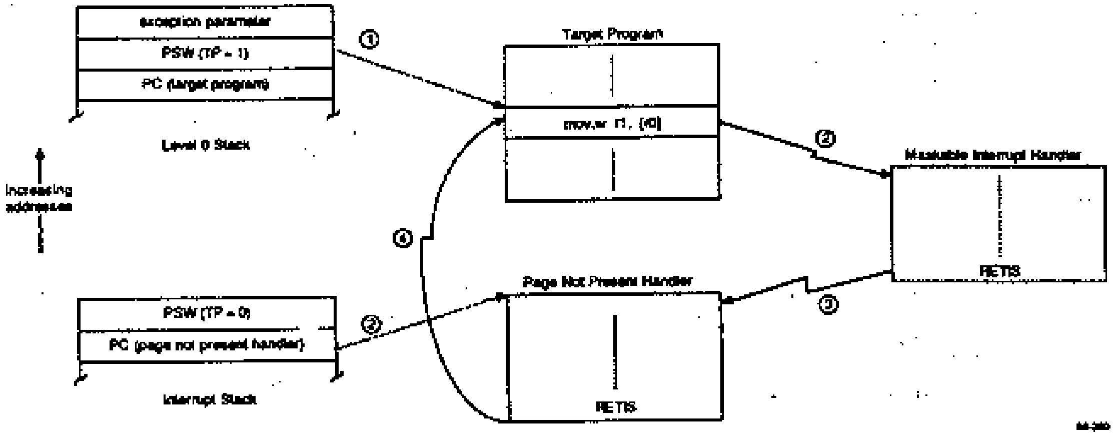


This example demonstrates the restart of an instruction following a page not present exception and delay of the instruction trace exception.

1. When an exception requiring the restart of the instruction and an instruction trace exception occur simultaneously, the instruction trace exception is ignored. The memory management exception frame is pushed on the level 0 stack with the TP field in the PSW field cleared and the PC image pointing to the first byte of the instruction. Program control is then transferred to the memory management exception handler.

2. When the page has been brought back into physical memory, the handler executes an RETIS instruction. The restored PC returns program control to the faulted instruction and because the PSW.TP flag is clear, no instruction trace exception will occur.

Figure 8–9 Simultaneous Page Not Present and Maskable Interrupt Exception Processing

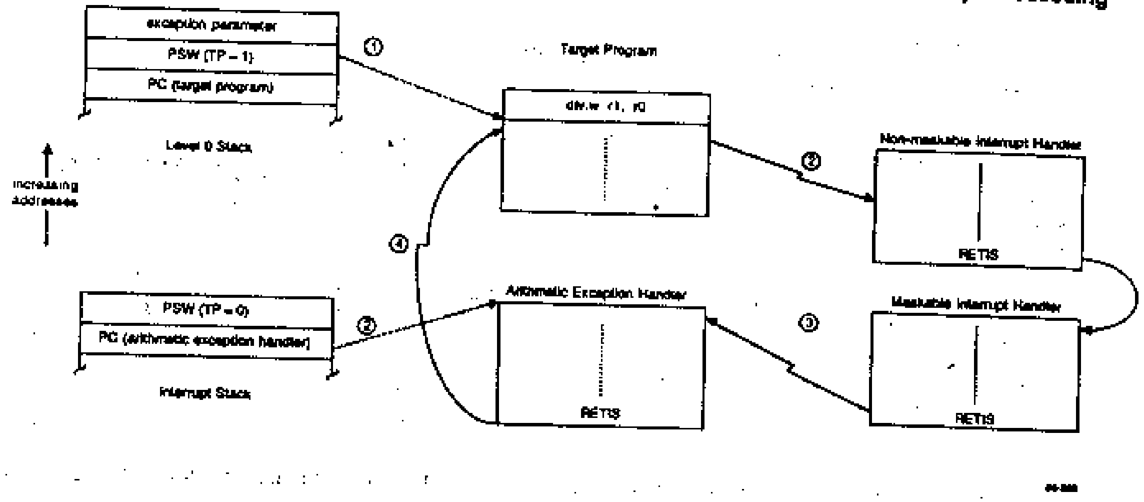


This example shows how the memory management exception is ignored if a simultaneous maskable interrupt occurs.

1. When a page not present exception and a maskable interrupt occur simultaneously, the memory management exception is ignored and only the maskable interrupt is serviced. The maskable interrupt frame is pushed on the interrupt stack with the TP field of the PSW cleared. Control is then transferred to the maskable interrupt handler with further maskable interrupts disabled.

2. The maskable interrupt handler terminates by executing a RETIS instruction, the restored PC and PSW.TP indicate to restart the instruction that originally caused the page not present exception. The re-execution of the instruction will again generate a page not present exception which will be processed as normal memory management exception.

Figure 8–10 Simultaneous Arithmetic Exception, Maskable and Non-Maskable Interrupt Processing



The final example demonstrates the occurrence of a simultaneous arithmetic exception with both maskable and non-maskable interrupts.

1. The exception processing begins by stacking the arithmetic exception frame on the level 0 stack.

2. Next, when maskable and non-maskable interrupts occur simultaneously, the maskable interrupt is ignored and the non-maskable interrupt stack frame is is pushed on the interrupt stack and the NMI handler is entered. The return PC on the stack contains the entry point of the arithmetic exception handler. During the processing of the non-maskable interrupt, maskable interrupts and additional non-maskable interrupts are disable.

3. When the NMI handler terminates, an RETIS instruction is executed. However, rather than popping the exception frame from the interrupt stack and entering the arithmetic exception handler, program control is vectored immediately to the specified maskable interrupt handler.

4. When the maskable interrupt handler terminates, the restored PC transfers control to the arithmetic exception handler. Following the correction of the exception condition, control is returned to the original program.

Interrupt/Exception Stacks

The active stack is determined by the contents of the EL (Execution Level) and IS (Interrupt Stack) fields in the PSW register as shown below:

EL	IS	Selected Stack
00	1	Interrupt Stack (ISP)
00	0	Level 0 Stack (L0SP)
01	0	Level 1 Stack (L1SP)
10	0	Level 2 Stack (L2SP)
11	0	Level 3 Stack (L3SP)

A program always sees only a single stack pointer regardless of the execution level. The PSW.EL and PSW.IS fields identify the active stack pointer (SP) and the contents of the active SP point to the top of stack (TOS) element on the ive stack.

The cache of five stack pointer registers (L0SP–L3SP, ISP) are independent of the user SP (R31). The user stack is switched whenever the execution level changes due to an interrupt or exception. Before changing stacks, the contents of the SP register are saved back in the contents of the corresponding stack pointer register. Because the SP is updated by any instruction affecting R31, the contents of the SP and the associated stack pointer register can differ during the execution of a program.

Notes

It is possible for the active stack to become invalid due to a memory management exception. the occurrence of this condition is described below:

(1) Level 1 / 2 / 3 Stacks Invalid

If the level 1 / 2 / 3 stack becomes invalid, an exception is generated and the execution level changes to level 0 and the level 0 stack becomes the active stack. Following the correction of the exception condition the original stack is restored.

If the stack invalid exception occurs during the stacking of exception information from an asynchronous task trap or change level exception, a double stack exception will occur.

(2) Level 0 Stack Invalid

If the level 0 stack is invalid, a level 0 stack invalid exception is generated and the interrupt stack (ISP) becomes the active stack.

(3) Interrupt Stack Invalid

If the interrupt stack is found to be invalid, the μPD70616 will halt until reset. For this reason, the interrupt stack must remain present.

Section 9
Software Debug Support

This section describes the software debug facilities of the μPD70616 microprocessor. The size and complexity of 32-bit systems demands that on-chip hardware contribute to the problem of debugging complex application and system software The on-chip software debug support offered by the μPD70616 aids in quickly identifying the errant sections of a program without spending hours over listings and real-time traces

Three separate software debug tools covering a wide range of debug strategies are supported.

- instruction trace
- instruction breakpoints
- address traps

Instruction trace is used to slowly execute a program to permit close observation of its behavior. Following the execution of each instruction, the program is interrupted and the instruction trace exception handler is entered. ecause of the coupling to the μPD70616 instruction set, instruction trace is frequently used to observe and debug programs at the assembly language level.

Often millions of instructions must be executed to reach the point of interest in a program and single stepping through each instruction is obviously inadequate. Instruction breakpoints allow real-time execution up to a special instruction and then cause an exception, allowing the breakpoint handler to regain control of the system and display the program state.

Both instruction trace and instruction breakpoints address the problems of debugging the instruction stream but are no help in debugging a section of ROM code or identifying wayward data write operations that can potentially destroy programs. To aid in the debug of these situations, a new type of hardware debug support known as address trapping is required. Address traps are the most flexible debugging tool since they combine an address along with an access type, permitting the hardware to distinguish between read, write and execute accesses anywhere within the virtual address space and to trap only those accesses which meet the specified trap conditions.

The level of on-chip hardware support for these three debug operations has been hitherto unavailable on any 8-, 16- or 32-bit microprocessor. This high level of software debug support allows a software debugging tool to perform many of the tasks previously relegated to expensive in-circuit emulators. The end result is a powerful, low cost means of designing complex software systems without sacrificing software reliability.

Instruction Trace

Instruction trace (also referred to as single stepping) is a tool used to force an exception following the execution of each instruction. Instruction trace allows a software engineer to observe the execution of a program at the instruction level to locate and correct software errors.

Instruction Trace Control

Instruction trace exceptions are controlled by the TE (Trace Enable) field in the PSW register.

PSW.TE = 0	instruction trace disabled
PSW.TE = 1	instruction trace enabled

Because this field in the upper halfword of the PSW, a privileged instruction is required to enable and disable instruction trace.

Instruction Trace Operation

When instruction trace exceptions are enabled, following the execution of each instruction an instruction trace exception occurs. Prior to entry into the instruction trace exception handler, the PC and PSW are pushed onto the level 0 stack and the PSW.TE field is cleared to allow the debugger to analyze program execution. To return back to the target program, the RETIS (Return from Interrupt – System) instruction is used.

Following the occurrence of an instruction trace exception, the level 0 stack contains the following information:

- PC of the next instruction
- exception code (0C00H)
- PSW image
- parameter count

Refer to section 8 for the organization of the level 0 stack following an instruction trace exception.

Instruction Trace Pending

As described in section 8, the existence of an interrupt or exception condition is checked following the execution of each instruction. Because instruction trace is the lowest priority of the many exceptions that can occur, all other higher priority exceptions will be processed ahead of an instruction trace exception.

The other exceptions that can occur can be classified into those which restart or resume instruction execution and those which occur after the execution of the instruction is completed. In the latter case, operation of instruction trace is as described following the correction of the higher priority exception condition. The former case presents a problem since the instruction has yet to execute and it would be incorrect to process the instruction trace exception after the higher priority exception condition was corrected and once again after the instruction completes execution.

The solution to this problem is for the µPD70616 to keep track of the exception type so to be able to restart the instruction and delay the instruction trace exception until the instruction has completed execution. This is accomplished by the TP (Trace Pending) field in the PSW register.

PSW.TP = 0	no instruction trace pending
PSW.TP = 1	instruction trace pending

When the PSW.TP field is cleared, an instruction trace exception is not pending and the instruction is restarted without servicing the instruction trace exception. If the PSW.TP field is set, either no other higher priority exception has occurred or the exception occurred after the execution of the instruction was completed and the instruction trace exception should be honored.

Consider the following examples. Suppose that a program is being debugged with the instruction trace mode enabled and a higher priority exception also occurs.

Case 1 Pre-execution exception

Assume that an instruction makes a memory reference to an non-existent page and causes a page fault. Because the instruction has not executed and must be restarted, the PSW.TP field is cleared prior to being pushed on the stack. After the exception information is placed on the stack, instruction trace is disabled and the memory management exception handler is entered. After the missing page has been brought into physical memory, the faulted instruction is restarted. Because the PSW.TP field is cleared, the instruction trace exception is delayed until the instruction completes execution.

Case 2 Post-execution exception

Assume that an integer overflow exception has occurred. Because the instruction has completed execution, the PSW.TP field is set prior to being pushed on the stack. After the exception information is placed on the stack, instruction trace is disabled and the integer overflow exception handler is entered. When the integer overflow handler is finished and executes a RETIS/RETIU instruction, the restored PSW is checked and because the PSW.TP field is set, an instruction trace exception occurs.

The PSW.TP field is controlled by the µPD70616 microprogram. When an exception occurs that requires the instruction to restarted or resumed, the TP field in the PSW image pushed onto the stack is cleared and the instruction trace exception is delayed until the instruction completes execution.

UPDPSW.W Instruction

Enable of instruction trace is generally by the RETIS instruction. It is also possible for the privileged UPDPSW.W instruction to modify the PSW.TE field and enable or disable instruction trace under the following circumstances:

- If instruction trace is enabled and a UPDPSW.W instruction executes and clears the PSW.TE field, a final instruction trace exception will occur after the completion of the UPDPSW.W instruction.
- If instruction trace is disabled and the execution of a UPDPSW.W instruction sets the PSW.TE field, the instruction trace exception will not occur until after the completion of the instruction following the UPDPSW.W instruction.

Instruction Trace Note

A final note on instruction trace operation. If the instruction trace exception handler accidentally enables instruction trace, an endless sequence of instruction trace exceptions can occur. Exercise caution when writing instruction trace exception handlers to avoid this situation.

Breakpoint Traps

The instruction breakpoint facility is used to implement program flow debugging[†]. An instruction breakpoint is set by replacing the first byte of an instruction with the one byte BRK instruction. The program then executes at full speed until the breakpoint is reached and executed causing the breakpoint trap.

Following the occurrence of an breakpoint trap, the level 0 stack contains the following information:

- PC of the current instruction
- exception code (0D00H)
- PSW image
- parameter count

Refer to section 8 for the organization of the level 0 stack following a breakpoint trap.

Note that only the first byte of a multi-byte instruction needs to be replaced with the BRK instruction. Subsequent bytes may be left unmodified without causing an exception since the breakpoint trap guarantees the remaining bytes of the instruction never reach the execution unit.

Also a note of caution concerning infinite breakpoint traps. The setting of an instruction breakpoint within the breakpoint exception handler will cause repeated breakpoint traps until some other exception occurs and terminates the sequence of instructions. Exercise caution to avoid this situation.

[†] For example, a high level language debugger can set breakpoint traps at the machine instructions corresponding to each line of the high level language source code.

Address Traps

Address traps are a powerful debugging facility that combine the occurrence of an address (or range of addresses) with one or more access types (read, write, execute) to generate an exception. Address traps differ from instruction trace and breakpoint traps since they operate on data accesses as well as instruction accesses and can span one or more tasks in a multitasking system.

Address Trap Operation

The setup of a µPD70616 address trap is performed as follows:

- address assignment
  Specifies the base address within which address traps will occur. For example, virtual address 10000H can be used as the base address.
- address mask assignment
  This parameter defines the range of addresses from the specified base address that address traps will occur. To trap accesses within the virtual page at address 10000H, the range is set to 4096 bytes.
- access type
  This parameter assigns one or more access types to the range of addresses. For instance, if traps are to occur on write access to the region, write access trapping must be specified.

In the example above, the address trap would be programmed to occur when the following conditions were all satisfied:

- virtual addresses 010000H–010FFFH
- write accesses

The µPD70616 allows two independent sets (address trap 0 and address trap 1) of address trap specifications to be defined simultaneously. Address trap specifications can be overlapped or use the same or different access types without restriction.

Address Trap Registers

Address traps are setup and controlled by three types of privileged registers and a field within the PSW register (refer to section 3 for detailed register specifications).

- Address Trap Enable (PSW.AE)
  The PSW.AE flag is used as a global enable and disable for the address trap logic.

PSW.AE = 0	address traps disabled
PSW.AE = 1	address traps enabled

  Because the AE flag resides in the privileged upper halfword of the PSW only operating system routines can control the operation of address traps.

- Address Trap Registers (ADTR0/ADTR1)
  The two ADTR registers each contain a 32-bit base address used to define a region by the trap logic. Register ADTR0 contains the base address for address trap 0 and ADTR1 contains the base address for address trap 1.

- Address Trap Mask Registers (ADTMR0/ADTMR1)
  The ADTMR registers are used to qualify the ADTR registers to define a range of addresses. Each of these registers contains a 32-bit mask value that identifies the corresponding bits of the ADTR registers as "don't care". ADTMR0 contains the mask value for ADTR0 and ADTMR1 contains the mask value for ADTR1.

  A limitation applies to the contents of the ADTMR registers. The low order two bits of these registers must be set in order to operate properly. Failure to heed this restriction will result in UNPREDICTABLE operation.

- Trap Mode Register (TRMOD)
  The TRMOD register contains the access type specifiers for each set of ADTR/ADTMR registers. Fields within the TRMOD contain the following flags:

Address Trap 0	• read..............read access to the address causes an address trap 0 • write..............write access to the address causes an address trap 0 • execute.........execute access to the address causes an address trap 0
Address Trap 1	• read..............read access to the address causes an address trap 1 • write..............write access to the address causes an address trap 1 • execute.........execute access to the address causes an address trap 1

Each of the above fields is independently assignable allowing trap conditions such as on read/write or read/write/execute access to a region. Address traps 0 and 1 can be disabled by simply selecting the no access condition (i.e., all bits within an access field cleared) in the appropriate field in the TRMOD register.

Address Trap Setup

Setting up for address traps involves programming of the ADTR/ADTMR register pairs, the TRMOD register and enabling address traps in the PSW register. An address trap will occur whenever all of the following conditions have been met:

1. PSW.AE = 1 (address traps enabled)

Address Trap 0	Address Trap 1
2. any of the TRMOD R/W/E bits are set	2. any of the TRMOD R/W/E bits are set
3. a programmed access type occurs	3. a programmed access type occurs
4. an address region 'hit' occurs	4. an address region 'hit' occurs

An address 'hit' occurs when the following condition is satisfied:

[AccessAddress and not(AddressMask)] = [TrapAddress and not(AddressMask)]

For example, to trap on accesses within a 128 byte region, an ADTMR is programmed as follows:

Figure 9–1 ADTMR Setup (128 byte region)

```
31                                                                     0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 | 1 | 1
```

Since the low order two bits of the ADTMR registers must be set, the smallest trap region is four bytes (4B) while largest trap region (all ADTMR bits set) is four gigabytes (4GB).

Address Trap Generation

Address traps in the µPD70616 occur when any byte within an address trap region is accessed with a specified access type. The actual occurrence of an address trap is delayed until both the instruction and the access is completed as follows:

- An execute access address trap is delayed until the instruction is executed. Instructions within the address trap region which are prefetched but not executed do not cause address traps.
- A read access address trap occurs after the access to the operand has commenced and the read operation has completed.
- A write access address trap occurs after the access to the operand has commenced and the write operation has completed.

No priority is assigned in the event of address traps detected simultaneously by each set of address trap logic. An address trap handler can detect this condition by examining the exception code placed on the level 0 stack by the processor.

Virtual/Physical Mode Address Traps

Address traps can operate in both the virtual and physical modes of the µPD70616. Address traps in virtual mode utilize virtual addresses while address traps in physical mode utilize physical addresses.

When operated in the physical mode, the trap address is regarded as a physical address and the high order 8-bits of the ADTMR registers should be set to match the addresses output on the 24-bit address bus. Failure to mask the upper 8-bits will result in UNPREDICTABLE operation.

When virtual mode is enabled, the trap address is regarded as a virtual address. No capability is available for trapping of physical addresses while in the virtual mode.

Address Trap Stack Contents

When the conditions for an address trap are met, information is placed on the level 0 stack and control is transferred to the address trap handler. The following information is placed on the level 0 stack:

- address of the instruction causing the trap (CurrentPC)
- address of the next instruction (NextPC)
- PSW image
- address trap exception code

The CurrentPC is used by the exception handler to analyze the instruction which caused the address trap while the NextPC is used as the return address to continue program execution. After the PSW image is pushed on the stack, the AE field within the PSW is cleared to disable further address traps. The exception code contains information on which set of address trap logic detected the access as shown below:

Section 10
V20/V30 Emulation Mode

This section describes the operation of the V20/V30 emulation mode. The V20/V30 emulation mode is included in the µPD70616 in order to allow system designers to take advantage of the large installed base of 16-bit software while simultaneously providing an upgrade path to a high performance 32-bit architecture for new applications.

V20/V30 emulation allows the design of a 32-bit system that is object code compatible with software designed for the µPD70108/116 microprocessors yet offers higher performance. With the exception of some system level instructions which require software emulation, the entire V20/V30 instruction set is implemented. In addition to the full resources of the V20/V30 microprocessors, other enhancements of the µPD70616 such as memory management and software debug mechanisms are available for use in developing and porting 16-bit V20/V30 software.

The software used to complete the V20/V30 architecture is called a virtual machine monitor. In addition to providing the software handlers for emulation of privileged instructions and processing of exceptions, the virtual machine monitor is used complete the emulation of the original system environment. System emulation allows programs written for a different system to execute even if it contains references to physical memory and peripheral devices. Trapping of I/O instructions and use of the address trap facilities simplify and aid the design of system emulations.

Virtual and Physical Address Modes

Like µPD70616 native mode, V20/V30 emulation mode operates in either virtual mode or physical mode depending on the state of the VM flag in the System Control Word (SYCW).

In the physical address mode, the 1MB emulation mode address space is mapped on to the lower 1MB of the 16MB native mode memory address space. Likewise, the V20/V30 64KB I/O address space is mapped into the lower 64K bytes of the µPD70616 16MB I/O address space. Because there is no address translation in physical mode, the protection mechanisms are disabled and only a single emulation context can be installed at a time.

Virtual mode V20/V30 emulation provides a number of advantages. With the address translation and protection mechanisms enabled, any number of native and emulation mode contexts can co-exist without interference. Virtual mode emulation mode tasks use the lower 1MB of the 4GB virtual address space but now the operating system through the address translation tables separates and protects each task and the demand paging mechanisms allow the utilization of secondary storage for complex multi-tasking systems.

In virtual mode, all emulation mode tasks execute at execution level 3. An emulation mode program has all of the facilities of a similar native mode program except that the address space is restricted. In particular, this applies to mapping of virtual address on to the I/O address space. Like the native mode instruction set, all emulation mode instructions are restartable or in the case of the emulation mode block transfer instructions, are interruptable and resumable.

Emulation Mode

The operating mode of the µPD70616 microprocessor is controlled by the EM flag in the PSW register. During native mode operation, the EM flag is cleared and the full set of µPD70616 resources are available to programs. Setting of the EM bit changes the processor mode to V20/V30 emulation mode and allows the execution of V20/V30 programs in a demand paged, protected environment.

Figure 9–2 Address Trap Exception Code Format

31							24	23						17	16	15 0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	b	a	number

bits 0:15	number	this field contains the number of bytes placed on the stack

bit 16	a	this field contains a flag which indicates if an address trap 0 occurred
		a = 0 no address trap 0 occurred
		a = 1 address trap 0 occurred

bit 17	b	this field contains a flag which indicates if an address trap 1 occurred
		b = 0 no address trap 1 occurred
		b = 1 address trap 1 occurred

bits 18:23	0	reserved field with each bit cleared
bits 24:31	'0E'H	address trap exception code

Refer to section 8 for details of the level 0 stack contents following an address trap.

Address Trap Notes

The following are notes on the operation of the µPD70616 address trap facilities.

Memory Indirect Addressing Modes

Address traps can occur during the effective address calculation of memory indirect addressing modes. If a memory indirect addressing mode is used and during the effective address calculation the access of the memory resident pointer is within an address trap region with read access enabled, an address trap will occur.

Instruction Trace Operation

It is possible to use the address trap logic to implement a second form of instruction trace. By setting the entire address space as the trap region and selecting the execute access type, an address trap will occur after the execution of each instruction.

Infinite Address Traps

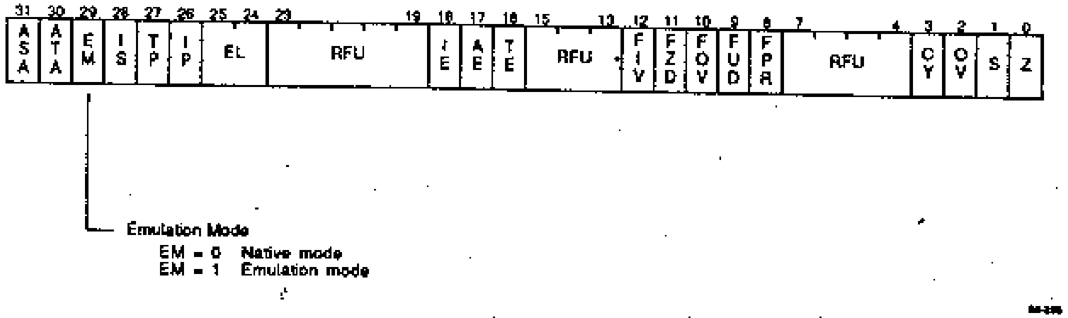
Although further address traps are automatically disabled following entry into the address trap handler, an endless series of address traps can occur if the trap handler re-enables address traps and a memory access meets the programmed trap conditions.

TLBNF Accesses

The address trap logic is disabled during address translation and system base table accesses. The internal µPD70616 microprogram will ignore these accesses even if address traps are programmed.

The EM flag is in a privileged field in the upper halfword of the PSW and only software running at execution level 0 can change processor modes. Because both the flag and the instruction streams must change simultaneously, the privileged RETIS instruction is used to enter V20/V30 emulation mode.

Figure 10–1. PSW Emulation Mode Specification

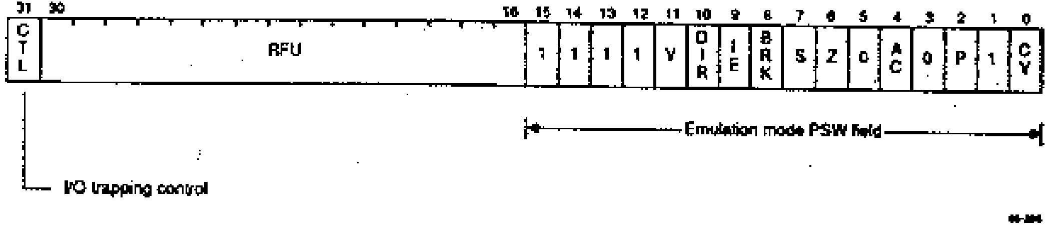


Program Status Word (PSW2)

Emulation and native mode utilize independent program status words which are switched automatically when emulation mode is entered. Separate PSW registers allows the virtual machine emulation to maintain a PSW image (such as maskable interrupt status) without having to physically mask or unmask interrupts.

While the PSW2 register and V20/V30 PSW appear similar, there are minor differences in the operation of the IE and BRK flags. In emulation mode, these flags can be in either the set or cleared state but they have no affect on the operation of the system since maskable interrupts and single stepping are controlled by fields in the native mode PSW. When an instruction is executed that attempts to change the state of either flag, an exception is generated. This permits the virtual machine software to intervene and provide a proper emulation of the system function in the µPD70616 system.

Figure 10–2. Emulation Mode PSW (PSW2)



bit 0 CY The CY (carry) flag indicates if a carry or borrow was generated as a result of the operation.

CY = 0 no carry (borrow) generated
CY = 1 carry (borrow) was generated

bit 1 1 Must be set

bit 2 P The P (parity) flag indicates the parity of the lower 8-bits of the result.

P = 0 odd parity
P = 1 even parity

bit 3 0 Must be cleared

bit 4 AC The AC (auxiliary carry) flag indicates if a carry was generated from the lower nibble to the upper nibble.

AC = 0 no carry
AC = 1 carry

bit 5 0 Must be cleared

bit 6 Z The Z (zero) flag indicates if the results of the operation were zero.

Z = 0 result is non-zero
Z = 1 result is zero

bit 7 S The S (sign) flag indicates if the results are negative (signed) or if the MSB is set (unsigned).

S = 0 result is positive or zero or MSB is 0
S = 1 result is negative or MSB is set

bit 8 BRK The BRK (break) flag contains the current single step status for emulation mode programs. It has no effect on single step operation and must be maintained by the native mode virtual machine software.

bit 9 IE The IE (interrupt enable) flag contains the current maskable interrupt status for emulation mode programs. It has no effect on maskable interrupts and must be maintained by the native mode virtual machine software.

bit 10 DIR The DIR (direction) flag indicates determines the direction of block transfer instructions.

DIR = 0 incrementing addresses
DIR = 1 decrementing addresses

bit 11 V The V (overflow) flag indicates if an overflow occurred.

V = 0 no overflow
V = 1 overflow

bits 12:15 1 Must be set

bits 16:30 RFU Reserved for future use

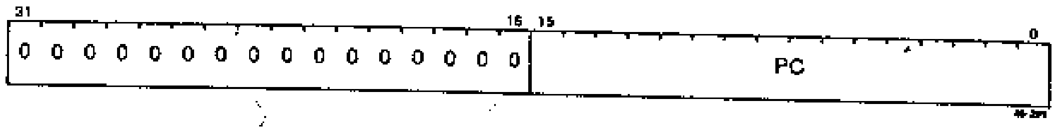
bit 31 CTL The CTL (I/O control) permits selective execution of emulation mode IN and OUT instructions without generation of an exception.

CTL = 0 I/O emulation enabled
CTL = 1 I/O emulation disabled

Program Counter (PC)

The emulation mode PC uses the low order 16-bits of the native mode program counter. Because the address space is restricted to the lower 1MB of the µPD70616 address space, the high order halfword must be zero.

Figure 10–3. Emulation Mode Program Counter



bits 0:15 PC The PC (program counter) field contains the 16-bit emulation mode program counter

bits 16:31 0 Must be zero

I/O Emulation Option

The I/O emulation option permits emulation mode programs to override the privileged status of IN and OUT instructions and permit selective execution of these I/O instructions without the generation of an exception. This option is controlled by the CTL bit in the PSW2 register as follows:

- CTL = 0 I/O emulation enabled (I/O instructions privileged)
- CTL = 1 I/O emulation disabled

Control of the trapping of I/O instructions is particularly useful in the design of virtual machine monitors for device drivers and other software with real-time I/O requirements.

Register Allocation

The emulation mode register set uses a subset of the native mode register set with the exception of the emulation mode PSW which is a physically distinct hardware resource. Since all V20/V30 registers are 16-bits in length, emulation mode registers occupy the lower halfword of each corresponding 32-bit native mode register.

The emulation mode register set is mapped onto native mode registers R0 through R11. The upper halfword of registers R0–R7 are unmodified by emulation mode programs. Since the V20/V30 microprocessors use a segmentation scheme with the segment registers used in the virtual address generation, the upper halfwords of the four 16-bit segment registers (R8–R11) are significant and must be zero.

In addition to the emulation mode register set, native mode registers R12 through R16 are used by the emulation facility as working storage and must not be modified by native mode programs. A diagram of the emulation mode register set is shown below.

Figure 10–4. Emulation Mode Register Set

Native Mode	Emulation Mode (31–0)
R31 (SP)	Not used
R30 (FP)	Not used
R29 (AP)	Not used
R28	Not used
R27	Not used
R26	Not used
R25	Not used
R24	Not used
R23	Not used
R22	Not used
R21	Not used
R20	Not used
R19	Not used
R18	Not used
R17	Not used
R16	Work register
R15	Work register
R14	Work register
R13	Work register
R12	Work register
R11	DS0
R10	SS
R9	PS
R8	DS1
R7	IY
R6	IX
R5	BP
R4	SP
R3	BW
R2	DW
R1	CW
R0	AW

	31–0
PSW	Not used
PC	PC
PSW2	PSW

Emulation Mode Instruction Set

The emulation mode instruction set includes nearly all of the V20/V30 instructions with the exception of system control instructions. The difference between the V20/V30 microprocessors and the V20/V30 emulation mode is that the system control instructions are considered to be privileged and are trapped and emulated by system software. This allows the µPD70616 system software to accommodate programs containing these instructions on a program by program basis without compromising the security of the operating system.

These privileged instructions include functions such as input/output, maskable interrupt control and instruction trace (single stepping). Instructions in these classes must be emulated relative to a particular system configuration in order to function properly.

The one exception is for simple I/O instructions. The system programmer has the option of not trapping IN and OUT instructions and instead allowing them to execute normally. This adds flexibility by allowing programs with real-time I/O constraints to function properly albeit in a system with less than full protection. The I/O option is enabled by the CTL bit in the PSW2 register.

Note that the V20/V30 µPD8080 emulation mode is *not* supported.

Instruction Set Summary

Below is a summary of the V20/V30 instruction set. For details of the instruction and architecture, refer to the µPD70108/116 User's Manual.

The following are descriptions of the instruction and conditions which cause exceptions when executed during emulation mode.

- Unconditional Instruction Exception

  Attempted execution of these instructions always causes an Emulation Mode Privileged Instruction exception to occur.

Trap instructions	BRK, BRK 3
Interrupt Processing	RETI
I/O Instructions	INM, OUTM
Processor Control Instructions	HALT, EI, DI

- Conditional Instruction Exceptions

  Conditional instructions may or may not cause an Emulation Mode Privileged Instruction exception depending on the current processor state.

  The V20/V30 IN and OUT instructions cause an exception if the CTL bit in the PSW2 register is cleared. The IN and OUT instructions execute normally when the CTL bit is set (I/O instruction trap enabled).

I/O Instructions	IN, OUT

  The POP PSW instruction will conditionally trap if a change of state to the IE and BRK bit positions will occur. No trap will occur if the state of these bit fields will remain unchanged.

PSW Instruction	POP PSW

Table 10–1. Emulation Mode Instruction Set

Instruction Class	Instructions
Data Transfer Instructions	MOV, MOV AH, PSW, MOV PSW, AH, XCH, CVTBW, CVTWL
Address Calculation Instructions	LDEA
Arithmetic Instructions	ADD, ADDC, SUB, SUBC, INC, DEC, MULU, MUL, DIVU, DIV, NEG
Comparison Instructions	CMP, TEST
Logical Instructions	NOT, AND, OR, XOR
Bit Field Instructions	INS, EXT
Bit Manipulation Instructions	TEST1, SET1, CLR1, NOT1, SET1/CLR1/NOT1 CY, SET1/CLR1 DIR
Shift Instructions	SHL, SHR, SHRA
Rotate Instructions	ROL, ROR, ROLC, RORC
BCD Adjust Instructions	ADJBA, ADJ4A, ADJBS, ADJ4S
BCD Conversion Instructions	CVTBD, CVTDB
Decimal String Instructions	ADD4S, SUB4S, CMP4S, ROL4, ROR4
Stack Instructions	PUSH, POP, PUSH R, POP R, PUSH PSW, POP PSW
Stack Frame Instructions	PREPARE, DISPOSE
Control Transfer Instructions	CALL, RET,Bcc, DBcc, BCWZ
Block Transfer Instructions	MOVBK, CMPBK, CMPM, LDM, STM
Translate Instruction	TRANS
Interrupt/Exception Return	RETI
Trap Instructions	BRK, BRK 3, BRKV
I/O Instructions	IN, OUT, INM, OUTM
Processor Control Instructions	HALT, EI, DI, BUSLOCK
Repeat Prefixes	REPC:, REPNC:, REP:, REPE:, REPZ:, REPNE:, REPNZ:
Segment Override Prefixes	DS0:, DS1:, SS:, PS:
Coprocessor Instructions	POLL, FPO1, FPO2
Miscellaneous Instructions	NOP, CHKIND

- Run-time Exceptions

  The following group of V20/V30 conditions are detected by emulation mode and cause an Emulation Mode Fault exception.

  – zero divide exception
  – single step trap
  – overflow exception
  – CHKIND instruction exception
  – undefined opcode

- Disregarded Instructions

  The following instructions are ignored:

Processor Control Instruction	BUSLOCK
Coprocessor Instruction	POLL

- System Implementation Exceptions

  The following V20/V30 instructions will cause an Emulation Mode Privileged Instruction exception depending on the system implementation.

  Coprocessor Instructions FPO1, FPO2

Table 10–2. Emulation Mode Privileged Operations

Instruction Class	Instructions
Reserved Instruction Exception Trap	INM, OUTM, HALT, EI, DI, BRK, BRK 3, RETI, MOV PSW, AH Zero Divide Single Step Overflow Exception CHKIND Exception Undefined Opcode
I/O Emulation Option	IN, OUT
System Dependent	FPO1, FP02
Data Dependent	POP PSW
Disregarded	BUSLOCK, POLL

Mode Transitions

A transition from native mode to emulation mode occurs when the EM bit in the PSW register is set when the PSW is restored during the execution of a RETIS instruction. Transitions from emulation mode to native mode occur as a result an interrupt or exception.

Native Mode → Emulation Mode

Two instances of native to emulation mode transitions exist. The initial entry into emulation mode is a four step process.

1. initialize the PS register (R9) and any other emulation mode registers
2. prepare a PSW image with EM = 1 and EL = 11 and push it onto the stack
3. push the initial PC of the emulation mode program on the stack
4. execute a RETIS instruction

The emulation mode PS register must be initialized by the native mode program since fetching of the initial emulation mode instruction requires valid program segment register contents. Other registers can be pre-initialized in native mode or initialized by the emulation mode program. Only the lower 16-bits of the emulation mode PC is valid and the upper halfword is ignored by µPD70616 emulation mode programs.

Emulation mode program execution can only take place at execution level 3. The EL field in the new PSW image is checked and if an execution level other than level 3 is specified an Illegal Data Field exception will occur.

Emulation Mode → Native Mode

Return to native mode from emulation mode programs occurs as the result of an interrupt or exception.

- Interrupts

  All interrupts on the µPD70616 microprocessor are processed in native mode. Upon the detection of an interrupt, the emulation mode program will be suspended and native mode will be restored.

- Exceptions

  Emulation mode exceptions occur as a result of program execution and require processing by a native mode handler to correct a fault or take some other predetermined action.

In either case, the return to emulation mode after processing of the exception or interrupt is by the RETIS instruction.

Termination of an emulation program is accomplished by the reserved 63H opcode. Attempted execution of this opcode will cause the Reserved Emulation Mode Opcode exception to be generated along with a specific exception code identifying the exception as a terminate operation.

mory Address Space

The one megabyte (1MB) address space of the V20/V30 microprocessors is mapped into the native mode virtual address space (virtual mode) or the memory address space (physical mode). In both cases the emulation mode memory address space occupies the lower megabyte of the corresponding virtual or memory address space.

Each of the four general purpose registers (R8–R11) are split into two fields. The lower 16-bit field contain the four V20/V30 segment registers in the lower halfword and the upper halfword must be must be zeroed for proper operation.

Address Generation

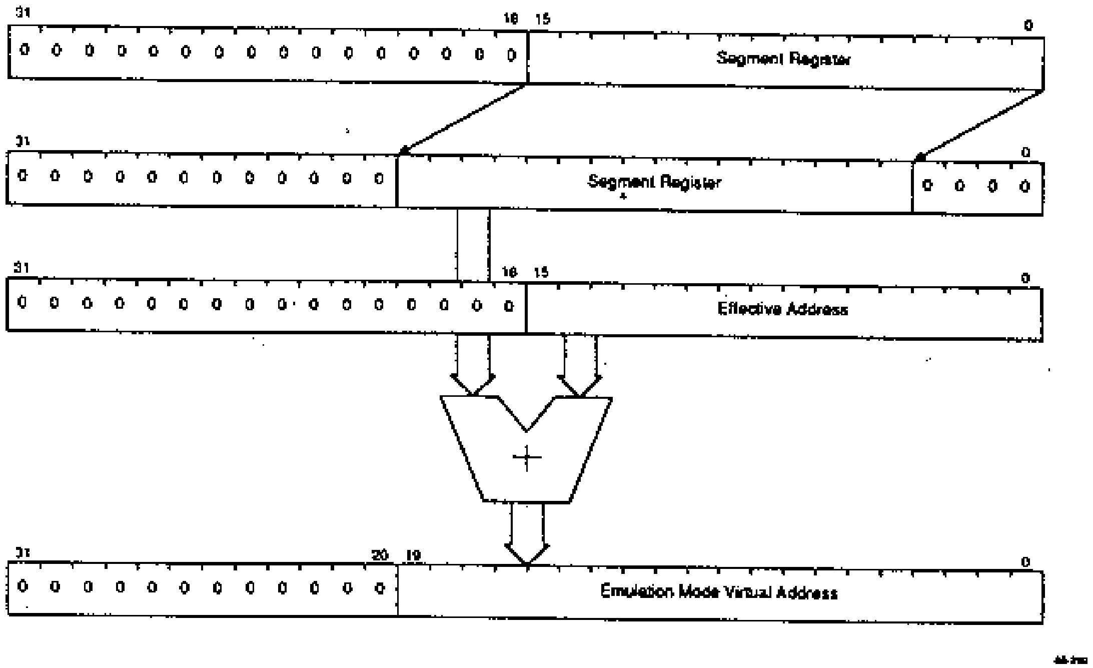
Both a segment register and an effective address are used to form the emulation mode address. The 16-bit effective address is first determined by the operand addressing mode and zero extended to 32-bit length. The 16-bit segment value in the appropriate segment register is left shifted four places and combined with the effective address to produce the 20-bit emulation mode address. This address is then zero extended to 32-bit length and either translated (virtual mode) or passed directly to the bus interface unit (physical mode).

Figure 10–5. Emulation Mode Segment Registers

31 — 16	15 — 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	DS1	R8
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	PS	R9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SS	R10
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	DS0	R11

bits 0:15	SEG	The SEG (segment) field contains the 16-bit segment base address
bits 16:31	RFU	Reserved for future use

Figure 10–6. Emulation Mode Address Generation



I/O Address Space

The V20/V30 emulation mode I/O address space (addresses 00000H to 0FFFFH) is mapped into the lower 64KB addresses of the µPD70616 address space if the emulation mode I/O option has been selected.

An exception will occur if an I/O instruction is attempted and the I/O emulation option is enabled.

Emulation Mode Notes

The µPD70616 V20/V30 emulation mode offers an upgrade path to a 32-bit environment while maintaining software compatibility with the previous generation of 16-bit devices. However, some precautions must be observed to guarantee the compatibility of 16-bit software in the 32-bit environment. These precautions relate to the small subset of application software which exhibit timing dependencies or utilize non-portable programming styles. A list of the programming restrictions follows.

- Instruction execution speed

  There is no correlation between the V20/V30 instruction execution times and the same instructions executed in V20/V30 emulation mode. There is no guarantee that programs dependent on the execution time of an instruction or sequence of instructions will execute correctly in V20/V30 emulation mode.

- Self modifying code

  Programs that modify the instruction stream and depend on the size of the V20/V30 instruction prefetch queue may not operate correctly on the µPD70616 microprocessor.

- Instruction length restriction

  The V20/V30 microprocessors place no limit on the length of an instruction. The µPD70616 microprocessor limits the length of a single instruction to 31 bytes. This situation can only arise out the use of duplicated instruction prefixes.

- µPD8080AF emulation mode

  The µPD8080AF emulation mode of the V20/V30 is not supported. The attempted execution of a BRKEM instruction will cause an illegal instruction exception .

- Undefined opcodes

  Attempted execution of undefined V20/V30 opcodes will cause an illegal instruction exception except in the following cases:

First Byte	Second Byte							
	MSB							LSB
C0H, C1H, D0H, D1H, D2H, D3H	X	X	1	1	0	X	X	X
F6H, F7H	X	X	0	0	1	X	X	X
FEH, FFH	X	X	1	1	1	X	X	X

  In the above three cases, results of the undefined instruction execution is UNPREDICTABLE.

- PUSH SP instruction

  The contents of the stack following the execution of a PUSH SP instruction differs between the V20/V30 and the µPD70616. The V20/V30 PUSH SP first decrements the SP register and then copies SP to the stack. The emulation mode PUSH SP instruction decrements SP by two and pushes the original value of the SP register on the stack.

- Address wrap-around

  Address wrap around occurs when a 16-bit or greater data access is made across a segment (64KB) or address space (1MB) boundary and is not supported by V20/V30 emulation mode. In the case where the operand wraps completely around, the address computation will be correct. Restrictions only apply to accesses across a segment or address space boundary.

  In the V20/V30 microprocessors, a 16-bit access to data located at offset 64K–1 from the base of the segment results in the least significant byte fetched from offset 64K–1 and the most significant byte fetched from offset zero within the same segment. Emulation mode differs in that 16-bit data is located at offset 64K–1 (LSB) and at offset 64K (MSB) in the same segment.

Section 11
Functional Redundancy Monitor

With each passing day, an increasing number of computer systems are called upon to perform in environments where the cost of an error in economic or human terms is extremely high and failures simply cannot be tolerated. Applications such as digital flight controls, telephone switching, transaction processing and others all require the use of systems which have a MTBF (mean time between failure) measured in the thousands or millions of years or whose availability (the time which the system is available to the application) exceeds 99% over a period of several years. The μPD70616 microprocessor is equipped with special purpose logic to aid in the design of highly reliable and fault tolerant computing systems.

Fault tolerant systems can be organized in many different configurations depending on the application and the degree of fault tolerance. The μPD70616 supports fault tolerant system design by permitting the use of multiple μPD70616 microprocessors operating as master/checker pairs. However, the design of a fault tolerant system requires much more than redundant hardware because software failures also contribute to the overall system failure rate. The combination of hardware fault coverage, fault isolation and system reconfiguration with the powerful exception handling facilities makes the μPD70616 an optimal solution for the design of high performance, fault tolerant systems.

Fault Tolerant System Configuration

Prior to a general discussion of fault tolerance and the issues concerning the design of highly reliable systems, a definition of the term should be presented. A system is said to be fault tolerant if it can survive and function in the presence of one or more faults. A fault tolerant system can further be classified as degrading or non-degrading. A system built using the concept of graceful degradation continues to operate as faults occur but the system successively reconfigures to a lower levels of performance. A non-degrading system uses massive redundancy in a way that failures are tolerated without a change in system performance until the redundancy is exhausted and the system ultimately fails.

All fault tolerant systems follow a similar path when confronted with a fault in the system. The basic sequence of steps include:

- fault detection
- fault isolation
- fault recovery
- system reconfiguration

Much emphasis must be placed on fault detection because the least reliable system is one with a latent or undetected fault that can propagate and corrupt the entire system. When a fault occurs it must be allowed to propagate in order to be detected. It is the task of a fault containment logic to detect as soon as possible the occurrence of a fault and contain or isolate it from the remaining healthy portion of the system. Next the fault must be analyzed to determine the extent of damage to the system and finally the system must reconfigure to remove the faulty section and restore to the system to a stable, healthy state.

One must also distinguish between two types of faults. The vast majority of faults that occur in a system can be characterized as transient faults. Transient faults are faults which disappear when re-tried because the disturbance which caused the fault is no longer present in the system. Because the frequency of transient faults is so much higher, a system cannot afford to degrade the system each time a fault is detected. Thus the capability of automatically re-trying the faulted operation without the intervention of the operating system is desirable.

Instructions which potentially cause a wrap around access violation are:

- 16-bit data transfers
- 32-bit data transfers
- bit field instructions
- save/restore multiple registers (PUSH R/POP R)
- stack frame allocation (PREPARE)

Segment wrap around restrictions also apply to the individual data transfers of 16-bit prefixed block transfer instructions.

- 64K, 1M boundary instruction fetch

  Instruction fetch across either a 64KB segment boundary or the 1MB memory address space boundary is prohibited.

- Memory allocation

  The memory and I/O address spaces for emulation programs is determined by the operating system. Caution should exercised in allocation of memory pages to emulation mode programs to prevent inadvertent exceptions due to improper permissions.

- Segment registers

  During emulation mode, if the upper order 16-bits of any segment register become non-zero, the results will be UNPREDICTABLE.

- Emulation mode single step

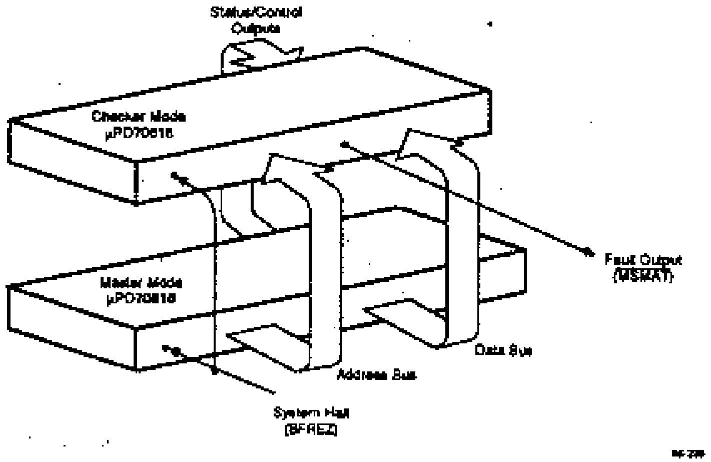
  Single stepping of emulation mode programs is possible using the µPD70616 single step facility to cause a trap upon entry into the emulation mode program. The emulation mode BRK flag has no effect on single step operation and must be emulated by the native mode single step trap handler.

- Emulation mode interrupt processing

  No provisions for the processing of interrupts or exceptions by emulation mode programs is provided. The IE flag in the emulation mode PSW has no meaning and the control of maskable interrupts must be done by the native mode exception handler.

Finally, the μPD70616 functional redundancy monitor is but the first step in the design of a fault tolerant system. There is no simple, one-step solution to design in fault tolerance since a system is only as reliable as the least reliable component. Thus, similar fault tolerant design techniques must also be applied to the memory and I/O subsystems in order to improve the reliability of these subsystems to the same level of reliability as the processor.

Figure 11–1: Functional Redundancy Monitoring



Functional Redundancy Monitor (FRM)

The concept of FRM is demonstrated by the duplex system shown in Figure 11–1. Functional redundancy monitoring is based on identical logic running in parallel with one μPD70616 microprocessor functioning as the master and the second μPD70616 configured as the checker. During operation, both the master and checker operate in lock-step on identical instruction and data streams but only the master actually drives the address, data and control buses. Instead of driving the bus outputs, the checker compares its independently computed state with the state asserted by the master μPD70616 and outputs a signal when a discrepancy is detected. The following is a description of the FRM mode.

1. Pin Description

   Four of the μPD70616 I/O pins are used for functional redundancy monitoring:

Table 11–1. FRM Pin Functions

Pin Name	I/O	Function
BMODE (FRM)	Input	Selects the normal or FRM operating mode
$\overline{\text{BLOCK}}$ ($\overline{\text{MSMAT}}$)	Output	Checker output indicating a mismatch has been detected
BFREZ	Input	Freezes processor operation
RT/$\overline{\text{EP}}$	Input	Continue/interrupt selection input

2. Mode Selection

   FRM is enabled and disabled by the BUSMD/FRM input pin. Refer to the μPD70616 data sheet for the technical details of enabling the FRM logic.

3. Monitoring

   When enabled, the checker microprocessor continuously compares the contents of the address bus, data bus, status outputs and MRQ* and UBE* pins with the expected values. Should a discrepancy be detected, the output will indicate a mismatch and either the master or checker processor could be faulty.

4. Fault Detection

   When a mismatch occurs, the checker μPD70616 will assert the MSMAT* (Mismatch) output. This signal in turn informs external logic that a fault has been detected and that intervention is required.

5. Fault Isolation

   In response to the mismatch signal, the external fault logic must stop both processors by asserting the BFREZ (Bus Freeze) input. When BFREZ is asserted, all further bus accesses are disabled and processing is suspended. In response to the BFREZ interrupt, the master and checker μPD70616 microprocessors will three-state the address, data and control buses. Both processors will remain isolated and disconnected from the system as long as BFREZ is asserted.

6. Fault Recovery

   Once the suspect devices have been isolated, the system can initiate the fault recovery procedures. Two methods of fault recovery are supported by the μPD70616, instruction continuation and interrupt.

   Instruction continuation involves restarting the processors as if no fault occurred. In this mode, the processors will start by re-trying the bus cycle that failed and continuing normal execution. The power of this mode is that the new master need not be the same as the old master, i.e., a reconfiguation can occur. The other alternative is the bus freeze interrupt which causes forces an interrupt and subsequent analysis by system software.

   Selection of the recovery mode is via the RT/EP* input pin. When the BFREZ input is negated, each μPD70616 checks the state of the RT/EP* pin. If RT/EP* is at a high level, then the instruction continuation mode is used. If at a low level, the bus freeze interrupt will occur and be serviced by both microprocessors.

Note that fault detection can only occur when an error induced by a fault propagates to an external pin. Because of the pipelined architecture and the fact that instructions can execute which modify the internal state with no external indication (such as register to register arithmetic operations), faults may not be detected until they are observable from the external pins.

Bus Freeze Interrupt

The bus freeze interrupt is used by external logic to force the processor into the bus freeze interrupt handler. Following the occurrence of a bus freeze interrupt, all interrupts are disabled and the PC and PSW are saved on the interrupt stack. The bus freeze interrupt handler is then expected to perform fault analysis of the system in an attempt to isolate the faulty component.

Because it may not be readily apparent which processor is faulty, the bus freeze interrupt handler in conjunction with the external voting logic will disconnect the suspect pair from the system and perform independent self tests on each processor by holding one processor in the bus freeze state while the other is tested in the normal mode. If either processor is found to be faulty, the system can be informed. Otherwise, the fault can be considered a transient fault with the processors being restarted and placed back on-line.

FRM Applications

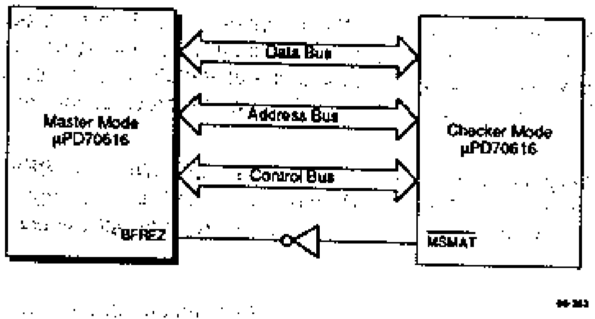
The design of a fault tolerant system can take many approaches depending on the performance, reliability and fault tolerance requirements for the proposed system. Two examples of typical fault tolerant system implementations are introduced but keep in mind that each application poses its own unique set of requirements.

With this in mind, a bibliography of selected works on fault tolerant system design is included at the end of this section. The descriptions of the µPD70616 fault detection and recovery capabilities and an general understanding of the overall requirements for fault tolerant system design will aid the designer in choosing the correct fault tolerant implementation for a particular application.

1. Duplex System

   A duplex system provides the basis for fault detection but provides no certain indication which unit has failed. While it is also possible under certain circumstances for a duplex system to isolate a fault to a specific unit though self test and reasonableness checks, there is no guarantee that the system will be able to survive the fault.

Figure 11–2. Duplex System Implementation



While limited in ability to recover from certain faults, dual systems do have varied applications. Many systems are required only to fail in a safe state. The specification of such a system would indicate what final state the system should end up in the event of a failure. The primary goal of a fail-safe system is for the state of the system not present any hazard to the user.

A typical example of a fail-safe system is the common traffic signal. Because town public works departments wish to minimize their capital outlays yet not endanger the driving populace, traffic signals are specified to be fail-safe. In the event of a failure, a traffic signal should never fail in the state that all lights are green since this would pose severe liabilities for the town. Instead, the final state for a failed traffic signal will typically be one set of lights flashing red and the other set of lights flashing yellow.

Because the cost of a system is directly proportional to the amount of redundancy, the duplex system is the system of choice for applications requiring to only be aware of faults in the system and are not required to recover completely from them.

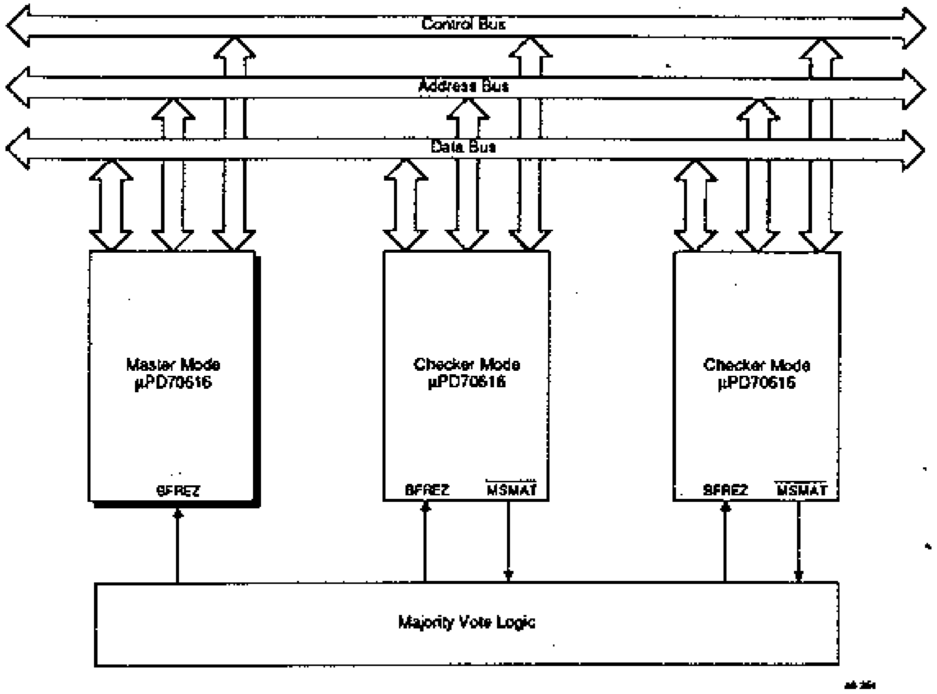
2. N-Modular Redundancy

   N-modular redundancy is a method used to improve upon the fault coverage of the duplex system to provide the capability to survive faults. In order to provide an indication of which processor in a system has failed, a majority vote of all processors in the system is required. An n-modular redundant system which uses an odd number of processors and majority voting logic to accurately determine the failed modules.

   The most common implementation of an n-modular redundant system is a TMR (Triple Modular Redundant) system. The TMR approach triplicates all logic and can use a 3-way majority voter to select the correct output and identify a faulty unit. Because a TMR system can survive the occurrence of any single fault, it is said to be 1-fault tolerant.

   In the event of a failure, the voting logic shuts down the failed unit, allowing the module to be repaired off-line while the system continues to function normally. After the repair has been completed, the module can be restored to the on-line condition and full fault tolerance restored.

Figure 11–3. Triple Modular Redundant System



Because the MTBF of a computing module is large (typically measured in months or years) and the MTTR (Mean Time To Repair) is generally 30 minutes to 1 hour, the probability of a second failure during the repair interval is low and the overall system MTBF can be on the order of hundreds of years.

Section 11 Bibliography

[1] A. Avizienis
Fault Tolerant Systems
IEEE Transactions on Computers, pp. 1304–1312, Vol. C–25, No. 12, December 1976

[2] D. P. Siewiorek and R. S. Swarz
The Theory and Practice of Reliable System Design
Digital Press, 1982

[3] G. D. Kraft and W. N. Toy
Microprogrammed.Control and Reliable Design of Small Computers
Prentice-Hall, 1981

[4] Special Issue on Fault Tolerant Computing
IEEE Computer, Vol. 17, No. 8, August 1984

[5] D. P. Siewiorek, C. G. Bell, and A. Newell
Fault Tolerant Systems (Section 6 of Part 2)
in Computer Structures : Principles and Examples
McGraw-Hill, 1982

Appendix A
Instruction Set Summary

This appendix contains a summary of the µPD70616 instruction set. The instructions are functionally grouped together by instruction type for ease of reference.

µPD70616 Program Register Set

Register	Description
R0 – R31	General purpose registers
PSW	Program Status Word
PC	Program Counter
AP	Argument Pointer (R29)
FP	Frame Pointer (R30)
SP	Stack Pointer (R31)

µPD70616 Privileged Register Set

Register	Description
ISP	Interrupt Stack Pointer
L0SP	Level 0 Stack Pointer
L1SP	Level 1 Stack Pointer
L2SP	Level 2 Stack Pointer
L3SP	Level 3 Stack Pointer
SBR	System Base Register
TR	Task Register
SYCW	System Control Word
TKCW	Task Control Word
PIR	Processor ID Register
PSW2	Program Status Word 2 (Emulation Mode PSW)
ATBR0	Area Table Base Register 0
ATLR0	Area Table Length Register 0
ATBR1	Area Table Base Register 1
ATLR1	Area Table Length Register 1
ATBR2	Area Table Base Register 2
ATLR2	Area Table Length Register 2
ATBR3	Area Table Base Register 3
ATLR3	Area Table Length Register 3
TRMOD	Trap Mode Register
ADTR0	Address Trap Register 0
ADTMR0	Address Trap Mask Register 0
ADTR1	Address Trap Register 1
ADTMR1	Address Trap Register 1

Definitions

Identifier	Description
reg	Any user register, R0–R31
.B	Byte integer or character (8-bit)
.H	Halfword integer or character (16-bit)
.W	Word integer (32-bit)
.D	Doubleword integer (64-bit)
.S	Short real (32-bit IEEE floating point value)
.L	Long real (64-bit IEEE floating point value)
.BH	Byte to halfword
.BW	Byte to word
.HW	Halfword to word
.HB	Halfword to byte
.WB	Word to byte
.WH	Word to halfword
.WS	Word to short real
.WL	Word to long real
.SW	Short real to word
.LW	Long real to word
.PZ	Packed to zoned decimal
.ZP	Zoned to packed decimal

PSW Condition Codes

Mnemonic	Flag Name
CY	Integer carry flag
OV	Integer overflow flag
S	Integer sign flag
Z	Integer zero flag
FIV	Floating point invalid operation flag
FZD	Floating point zero divide flag
FOV	Floating point overflow flag
FUD	Floating point underflow flag
FPR	Floating point precision flag

Flag Operations

Identifier	Description
(blank)	No change
–	No change
0	Cleared to 0
1	Set to 1
*	Set or cleared according to result
R	Restored to previous value

Opcode Fields

Condition Code Field

The PSW flags are used by the conditional branch instructions to determine the conditions necessary to transfer program control. These instructions include the Bcc (Branch on Condition), DBcc (Decrement and Branch on Condition) and TRAP instructions.

c3	c2	c1	c0	Name	Condition
0	0	0	0	Overflow	OV = 1
0	0	0	1	Not overflow	OV = 0
0	0	1	0	Carry/Lower	CY = 1
0	0	1	1	No carry/Not lower	CY = 0
0	1	0	0	Zero	Z = 1
0	1	0	1	Not zero	Z = 0
0	1	1	0	Not higher	$(CY \vee Z) = 1$
0	1	1	1	Higher	$(CY \vee Z) = 0$
1	0	0	0	Sign/Negative	S = 1
1	0	0	1	Not sign/Positive	S = 0
1	0	1	0	True	Always
1	0	1	1	False	Never
1	1	0	0	Less than	$(S \oplus OV) = 1$
1	1	0	1	Greater or equal	$(S \oplus OV) = 0$
1	1	1	0	Less or equal	$(S \oplus OV) \vee Z = 1$
1	1	1	1	Greater	$(S \oplus OV) \vee Z = 0$

Integer Size Field

The integer size field specifies the integer data type for integer instructions. The μPD70616 supports both signed and unsigned integers in each of the integer formats.

siz	Operand Size
00	Byte
01	Halfword
10	Word
11	Reserved

Floating Point Size Field

The floating point size field determines the operand data type for the floating point instructions. Both IEEE short and long real data types are supported by the μPD70616 microporcessor.

s	Operand Size
0	Short Real (32-bit)
1	Long Real (64-bit)

Character Size Field

The direction field is used by the character and bit strings instructions to determine the direction of string processing. The programmer has the option of processing character and bit strings in the upward (increasing addresses) or downward (decreasing addresses) direction to allow transfer and manipulation of overlapping strings with a single instruction.

c	Operand Size
0	Byte character
1	Halfword character

Direction Field

The direction field is used by the character and bit strings instructions to determine the direction of string processing. The programmer has the option of processing character and bit strings in the upward (increasing addresses) or downward (decreasing addresses) direction to allow transfer and manipulation of overlapping strings with a single instruction.

d	Direction
0	Upward (increasing addresses)
1	Downward (decreasing addresses)

Displacement Size Field

The displacement size field is found in the conditional branch instructions and determines whether an 8- or 16-bit displacment field follows the opcode.

b	Displacement Size
0	Byte (8-bit signed displacement)
1	Halfword (16-bit signed displacement)

Instruction Set Reference

	Opcode			Flags				
Mnemonic	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	Format	CY	OV	S	Z	Exceptions
			Data Transfer Instructions					
MOV.B	0 0 0 0 1 0 0 1		I, II					1, 3
MOV.H	0 0 0 1 1 0 1 1		I, II					1, 3
MOV.W	0 0 1 0 1 1 0 1		I, II					1, 3
MOV.D	0 0 1 1 1 1 1 1		I, II					1, 3
MOVS.BH	0 0 0 0 1 0 1 0		I, II					1
MOVS.BW	0 0 0 0 1 1 0 0		I, II					1
MOVS.HW	0 0 0 1 1 1 0 0		I, II					1
MOVZ.BH	0 0 0 0 1 0 1 1		I, II					1
MOVZ.BW	0 0 0 0 1 1 0 1		I, II					1
MOVZ.HW	0 0 0 1 1 1 0 1		I, II					1
MOVT.HB	0 0 0 1 1 0 0 1		I, II	–	•	–	–	1
MOVT.WB	0 0 1 0 1 0 0 1		I, II	–	•	–	–	1
MOVT.WH	0 0 1 0 1 0 1 1		I, II	–	•	–	–	1
XCH	0 1 0 0 0 siz 1		I, II					1, 3
MOVEA	0 1 0 0 0 siz 0		I, II					1
RVBYT	0 0 1 0 1 1 0 0		I, II					1
RVBIT	0 0 0 0 1 0 0 0		I, II					1
			Integer Arithmetic Instructions					
ADD	1 0 0 0 0 siz 0		I, II	•	•	•	•	1
ADDC	1 0 0 1 0 siz 0		I, II	•	•	•	•	1
SUB	1 0 1 0 1 siz 0		I, II	•	•	•	•	1
SUBC	1 0 0 1 1 siz 0		I, II	•	•	•	•	1
MUL	1 0 0 0 0 siz 1		I, II	–	•	•	•	1
MULU	1 0 0 1 0 siz 1		I, II	–	•	•	•	1
MULX	1 0 0 0 0 1 1 0		I, II	–	•	•	•	1
MULUX	1 0 0 1 0 1 1 0		I, II	–	•	•	•	1
DIV	1 0 1 0 0 siz 1		I, II	–	•	•	•	1, 4
DIVU	1 0 1 1 0 siz 1		I, II	–	0	•	•	1, 4
DIVX	1 0 1 0 0 1 1 0		I, II	–	•	•	•	1, 4
DIVUX	1 0 1 1 0 1 1 0		I, II	–	•	•	•	1, 4
REM	0 1 0 1 0 siz 0		I, II	–	0	•	•	1, 4
REMU	0 1 0 1 0 siz 1		I, II	–	0	•	•	1, 4
INC	1 1 0 1 1 siz -		III	•	•	•	•	1
DEC	1 1 0 1 0 siz -		III	•	•	•	•	1
NEG	0 0 1 1 1 siz 1		I, II	•	•	•	•	1
CMP	1 0 1 1 1 siz 0		I, II	•	•	•	•	
TEST	1 1 1 1 0 siz -		III	0	0	•	•	
			Logical Instructions					
AND	1 0 1 0 0 siz 0		I, II	–	0	•	•	1
OR	1 0 0 0 1 siz 0		I, II	–	0	•	•	1
XOR	1 0 1 1 0 siz 0		I, II	–	0	•	•	1
NOT	0 0 1 1 1 siz 0		I, II	–	0	•	•	1

Mnemonic	Opcode		Format	Flags				Exceptions
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		CY	OV	S	Z	
Shift/Rotate Instructions								
SHA	1 0 1 1 0 siz 1		I, II	•	•	•	•	1
SHL	1 0 1 0 0 siz 1		I, II	•	0	•	•	1
ROT	1 0 0 0 1 siz 1		I, II	•	0	•	•	1
ROTC	1 0 0 1 1 siz 1		I, II	•	0	•	•	1
Floating Point Instructions								
MOVF	0 1 0 1 1 1 s 0	0 0 0 0 1 0 0 0	II	•	0	•	•	1, 3, 7, 9
ADDF	0 1 0 1 1 1 s 0	0 0 0 1 1 0 0 0	II	•	0	•	•	1, 3, 6, 7, 8, 9
SUBF	0 1 0 1 1 1 s 0	0 0 0 1 1 0 0 1	II	•	0	•	•	1, 3, 6, 7, 8, 9
MULF	0 1 0 1 1 1 s 0	0 0 0 1 1 0 1 0	II	•	0	•	•	1, 3, 6, 7, 8, 9
DIVF	0 1 0 1 1 1 s 0	0 0 0 1 1 0 1 1	II	•	0	•	•	1, 3, 6, 7, 8, 9, 10, 11
CMPF	0 1 0 1 1 1 s 0	0 0 0 0 0 0 0 0	II	•	•	•	•	1, 3, 6, 7, 8, 9
NEGF	0 1 0 1 1 1 s 0	0 0 0 0 1 0 0 1	II	•	0	•	•	1, 3, 7, 9
ABSF	0 1 0 1 1 1 s 0	0 0 0 0 1 0 1 0	II	0	0	0	•	1, 3, 7, 9
SCLF	0 1 0 1 1 1 s 0	0 0 0 1 0 0 0 0	II	•	0	•	•	1, 3, 6, 7, 8, 9
CVTF	0 1 0 1 1 1 1 1	0 0 0 0 1 0 0 0	II	•	0	•	•	1, 3, 6, 7, 8, 9
CVT.WS	0 1 0 1 1 1 1 1	0 0 0 0 0 0 0 0	II	•	0	•	•	1, 3, 6, 7, 8, 9
CVT.WL	0 1 0 1 1 1 1 1	0 0 0 1 0 0 0 1	II	•	0	•	•	1,3, 6, 7, 8, 9
CVT.SW	0 1 0 1 1 1 1 1	0 0 0 0 0 0 0 1	II	–	•	•	•	1, 3, 6, 7, 8, 9
CVT.LW	0 1 0 1 1 1 1 1	0 0 0 0 1 0 0 1	II	–	•	•	•	1, 3, 6, 7, 8, 9
TRAPFL	1 1 0 0 1 0 1 1		V					1, 3, 6, 7, 9
Decimal Arithmetic Instructions								
ADDDC	0 1 0 1 1 0 0 1	0 0 0 0 0 0 0 0	VIIc	•	–	–	•	1, 5
SUBDC	0 1 0 1 1 0 0 1	0 0 0 0 0 0 0 1	VIIc	•	–	–	•	1, 5
SUBRDC	0 1 0 1 1 0 0 1	0 0 0 0 0 0 1 0	VIIc	•	–	–	•	1, 5
CVTD.PZ	0 1 0 1 1 0 0 1	0 0 0 1 0 0 0 0	VIIc	–	–	–	•	1, 5
CVTD.ZP	0 1 0 1 1 0 0 1	0 0 0 1 1 0 0 0	VIIc	–	–	–	•	1, 5
Bit Manipulation Instructions								
TEST1	1 0 0 0 0 1 1 1		I, II	•	–	–	•	1, 2
SET1	1 0 0 1 0 1 1 1		I, II	•	–	–	•	1, 2
CLR1	1 0 1 0 0 1 1 1		I, II	•	–	–	•	1, 2
NOT1	1 0 1 1 0 1 1 1		I, II	•	–	–	•	1, 2
Bit Field Instructions								
EXTBFS	0 1 0 1 1 1 0 1	0 0 0 0 1 0 0 0	VIIb					1, 2
EXTBFZ	0 1 0 1 1 1 0 1	0 0 0 0 1 0 0 1	VIIb					1, 2
EXTBFL	0 1 0 1 1 1 0 1	0 0 0 0 1 0 1 0	VIIb					1, 2
INSBFR	0 1 0 1 1 1 0 1	0 0 0 1 1 0 0 0	VIIc					1, 2
INSBFL	0 1 0 1 1 1 0 1	0 0 0 1 1 0 0 1	VIIc					1, 2
CMPBFS	0 1 0 1 1 1 0 1	0 0 0 0 0 0 0 0	VIIb	•	•	•	•	1, 2
CMPBFZ	0 1 0 1 1 1 0 1	0 0 0 0 0 0 0 1	VIIb	•	•	•	•	1, 2
CMPBFL	0 1 0 1 1 1 0 1	0 0 0 0 0 0 1 0	VIIb	•	•	•	•	1, 2
Bit String Instructions								
MOVBS	0 1 0 1 1 0 1 1	0 0 0 0 1 0 0 d	VIIb					1
NOTBS	0 1 0 1 1 0 1 1	0 0 0 0 1 0 1 d	VIIb					1

Mnemonic	Opcode		Format	Flags				Exceptions
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0		CY	OV	S	Z	
Bit String Instructions (cont)								
ANDBS	0 1 0 1 1 0 1 1	0 0 0 1 0 0 0 d	VIIb					1
ANDNBS	0 1 0 1 1 0 1 1	0 0 0 1 0 0 1 d	VIIb					1
ORBS	0 1 0 1 1 0 1 1	0 0 0 1 0 1 0 d	VIIb					1
ORNBS	0 1 0 1 1 0 1 1	0 0 0 1 0 0 1 d	VIIb					1
XORBS	0 1 0 1 1 0 1 1	0 0 0 1 1 0 0 d	VIIb					1
XORNBS	0 1 0 1 1 0 1 1	0 0 0 1 0 0 1 d	VIIb					1
SCH0BS	0 1 0 1 1 0 1 1	0 0 0 0 0 0 0 d	VIIb					1
SCH1BS	0 1 0 1 1 0 1 1	0 0 0 0 1 0 1 d	VIIb					1
Character Manipulation Instructions								
MOVC	0 1 0 1 1 0 c 0	0 0 0 0 1 0 0 d	VIIa					1, 3
MOVCF	0 1 0 1 1 0 c 0	0 0 0 0 1 0 1 d	VIIa					1, 3
MOVCS	0 1 0 1 1 0 0 0	0 0 0 0 1 1 0 0	VIIa					1, 3
CMPC	0 1 0 1 1 0 c 0	0 0 0 0 0 0 0 0	VIIa	–	–	•	•	1, 3
CMPCF	0 1 0 1 1 0 c 0	0 0 0 0 0 0 0 1	VIIa	–	–	•	•	1, 3
CMPCS	0 1 0 1 1 0 c 0	0 0 0 0 0 0 1 0	VIIa	–	–	•	•	1, 3
SCHC	0 1 0 1 1 0 c 0	0 0 0 1 1 0 0 d	VIIb	–	–	–	•	1, 3
SKPC	0 1 0 1 1 0 c 0	0 0 0 1 1 0 1 d	VIIb	–	–	–	•	1, 3
Stack Manipulation Instructions								
PUSH	1 1 1 0 1 1 1 -		II					1, 3
POP	1 1 1 0 0 1 1 -		II					1, 3
PUSHM	1 1 1 0 1 1 0 -		II					1, 3
POPM	1 1 1 0 0 1 0 -		II	R	R	R	R	1, 3 (Note 1)
PREPARE	1 1 0 1 1 1 1 -		II					1, 3
DISPOSE	1 1 0 0 1 1 0 0		V					
Control Transfer Instructions								
BC	0 1 1 b 0 0 1 0		IV					
BE	0 1 1 b 0 1 0 0		IV					
BGE	0 1 1 b 1 1 0 1		IV					
BGT	0 1 1 b 1 1 1 1		IV					
BH	0 1 1 b 0 1 1 1		IV					
BL	0 1 1 b 0 0 1 0		IV					
BLE	0 1 1 b 1 1 1 0		IV					
BLT	0 1 1 b 1 1 0 0		IV					
BN	0 1 1 b 1 0 0 0		IV					
BNC	0 1 1 b 0 0 1 1		IV					
BNE	0 1 1 b 0 1 0 1		IV					
BNH	0 1 1 b 0 1 1 0		IV					
BNL	0 1 1 b 0 0 1 1		IV					
BNV	0 1 1 b 0 0 0 1		IV					
BNZ	0 1 1 b 0 1 0 1		IV					
BP	0 1 1 b 1 0 0 1		IV					
BR	0 1 1 b 1 0 1 0		IV					
DBC	1 1 0 0 0 1 1 0	reg 0 0 1	VI					
DBE	1 1 0 0 0 1 1 0	reg 0 1 0	VI					

	Opcode			Flags				
Mnemonic	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	Format	CY	OV	S	Z	Exceptions
Control Transfer Instructions (cont)								
DBGE	1 1 0 0 0 1 1 1	reg 1 1 0	VI					
DBGT	1 1 0 0 0 1 1 1	reg 1 1 1	VI					
DBH	1 1 0 0 0 1 1 1	reg 0 1 1	VI					
DBL	1 1 0 0 0 1 1 0	reg 0 0 1	VI					
DBLE	1 1 0 0 0 1 1 0	reg 1 1 1	VI					
DBLT	1 1 0 0 0 1 1 0	reg 1 1 0	VI					
DBN	1 1 0 0 0 1 1 0	reg 1 0 0	VI					
DBNC	1 1 0 0 0 1 1 1	reg 0 0 1	VI					
DBNE	1 1 0 0 0 1 1 1	reg 0 1 0	VI					
DBNH	1 1 0 0 0 1 1 0	reg 0 1 1	VI					
DBNL	1 1 0 0 0 1 1 1	reg 0 0 1	VI					
DBNV	1 1 0 0 0 1 1 1	reg 0 0 0	VI					
DBNZ	1 1 0 0 0 1 1 1	reg 0 1 0	VI					
DBP	1 1 0 0 0 1 1 1	reg 1 0 0	VI					
DBR	1 1 0 0 0 1 1 0	reg 1 0 1	VI					
DBV	1 1 0 0 0 1 1 0	reg 0 0 0	VI					
DBZ	1 1 0 0 0 1 1 0	reg 0 1 0	VI					
TB	1 1 0 0 0 1 1 1	reg 1 0 1	VI					
JMP	1 1 0 1 0 1 1 -		III					1
BSR	0 1 0 0 1 0 0 0		IV					
JSR	1 1 1 0 1 0 0 -		III					1
RSR	1 1 0 0 1 0 1 0		V					
CALL	0 1 0 0 1 0 0 1		I, II					1
RET	1 1 1 0 0 0 1 -		III					
BRK	1 1 0 0 1 0 0 0		V					
BRKV	1 1 0 0 1 0 0 1		V					
TRAP	1 1 1 0 1 0 0 -		III					
RETIU	1 1 1 0 1 0 1 -		III					
Miscellaneous Instructions								
NOP	1 1 0 0 1 1 0 1		V					
GETPSW	1 1 1 1 0 1 1 -		III					1
UPDPSW.H	0 1 0 0 1 0 1 0		I, II	•	•	•	•	
CHLVL	0 1 0 0 1 0 1 1		I, II					1
CHKAR	0 1 0 0 1 1 0 1		I, II	•	–	•	•	1
CHKAW	0 1 0 0 1 1 1 0		I, II	•	–	•	•	1
CHKAE	0 1 0 0 1 1 1 1		I, II	•	–	•	•	1
TASI	1 1 1 0 0 0 0 -		III	•	•	•	•	1
CAXI	0 1 0 0 1 1 0 0		I	•	•	•	•	1
SETF	0 1 0 0 0 1 1 1		I, II					1
Privileged Instructions								
LDPR	0 0 0 1 0 0 1 0		I, II					2, 12
STPR	0 0 0 0 0 0 1 0		I, II					1, 2, 12
CLRTLB	1 1 1 1 1 1 1 -		III					12
CLRTLBA	0 0 0 1 0 0 0 0		V					12

	Opcode			Flags				
Mnemonic	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	Format	CY	OV	S	Z	Exceptions
Privileged Instructions (cont)								
GETATE	0 0 0 0 0 1 0 1		I, II	–	–	–	•	1, 12
UPDATE	0 0 0 1 0 1 0 1		I, II	–	–	–	•	1, 12
GETPTE	0 0 0 0 0 1 0 0		I, II	•	–	–	•	1, 12
UPDPTE	0 0 0 1 0 1 0 0		I, II	•	–	–	•	12
GETRA	0 0 0 0 0 0 1 1		I, II	•	–	–	•	1, 12
IN	0 0 1 0 0 siz 0		I, II					1, 12
OUT	0 0 1 0 0 siz 1		I, II					1, 12
LDTASK	0 0 0 0 0 0 0 1		I, II					12
STTASK	1 1 1 1 1 1 0 -		III					12
RETIS	1 1 1 1 1 0 1 -		III	•	•	•	•	2, 12
UPDPSW.W	0 0 0 1 0 0 1 1		I, II	•	•	•	•	12
HALT	0 0 0 0 0 0 0 0		V					12

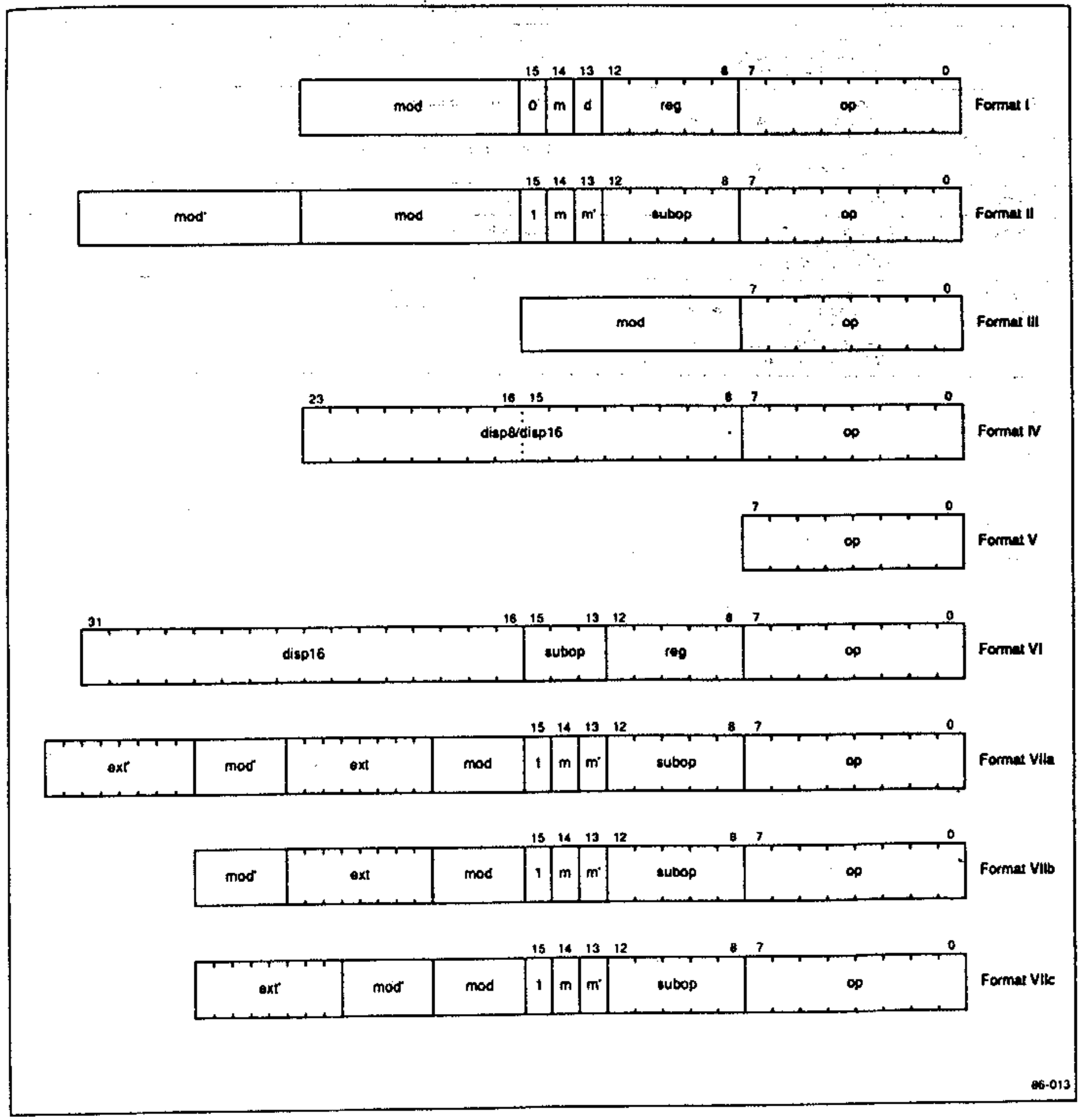
Exceptions

1. Illegal Addressing Mode
2. Illegal Data Type
3. Reserved Addressing Mode
4. Integer Zero Divide
5. Illegal Decimal Format
6. Floating Point Overflow
7. Floating Point Undeflow
8. Floating Point Precision
9. Reserved Floating Point Operand
10. Invalid Floating Point Operation
11. Floating Point Zero Divide
12. Privileged Instruction

Notes

1. Flags updated if PSW is specified in the register list

Appendix B Instruction Formats



Instruction Format Summary

Format	Fields	Description
Format I	Opcode Register Field Operand Addressing Mode	Fixed length data instructions using register/register and register/memory addressing modes
Format II	Opcode Operand 1 Addressing Mode Operand 2 Addressing Mode	Fixed length data instructions using memory/memory addressing modes and floating point instructions
Format III	Opcode Operand 1 Addressing Mode	Single operand instructions
Format IV	Opcode PC relative displacement	Conditional branch instructions
Format V	Opcode	Zero operand instructions
Format VI	Opcode Register Field PC relative displacement	Loop instructions
Format VII	Opcode Operand 1 Addressing Mode Operand 1 Length Operand 2 Addressing Mode Operand 2 Length	Variable length data instructions (character string, bit string, decimal arithmetic)

Appendix C
Addressing Mode Encodings

mod									m	Addressing Mode
71	63	55	47	39	31	23	15	7 … 0		
								011 Rn	1	Rn
								001 Rn	0	[Rn]
								100 Rn	1	[Rn+]
								101 Rn	1	[−Rn]
								1110 val	0	immed.4
							disp	000 Rn	0	disp.8[Rn]
							disp	11110000	0	disp.8[PC]
							disp	100 Rn	0	[disp.8[Rn]]
							disp	11111000	0	[disp.8[PC]]
							011 Rn	110 Rx	1	[Rn](Rx)
							val	11110100	0	immed.8
						disp		001 Rn	0	disp.16[Rn]
						disp		11110001	0	disp.16[PC]
						disp		101 Rn	0	[disp.16[Rn]]
						disp		11111001	0	[disp.16[PC]]
						val		11110100	0	immed.16
						disp1	disp2	000 Rn	1	disp1.8[disp2.8[Rn]]
						disp1	disp2	11111100	0	disp1.8[disp2.8[Rn]]
						disp	000 Rn	110 Rx	1	disp.8[Rn](Rx)
						disp	11110000	110 Rx	1	disp.8[PC](Rx)
						disp	100 Rn	110 Rx	1	[disp.8[Rn]](Rx)
						disp	11111000	110 Rx	1	[disp.8[PC]](Rx)
					disp		001 Rn	110 Rx	1	disp.16[Rn](Rx)
					disp		11110001	110 Rx	1	disp.16[PC](Rx)
					disp		100 Rn	110 Rx	1	[disp.16[Rn]](Rx)
					disp		11111000	110 Rx	1	[disp.16[PC]](Rx)
				disp				010 Rn	0	disp.32[Rn]
				disp				11110010	0	disp.32[PC]
				disp				110 Rn	0	[disp.32[Rn]]
				disp				11111010	0	[disp.32[PC]]
				disp1		disp2		001 Rn	1	disp1.16[disp2.16[Rn]]
				disp1		disp2		11111101	0	disp1.16[disp2.16[PC]]
				addr				11110011	0	/addr
				addr				11111011	0	[/addr]
				val				11110100	0	immed.32
			disp				010 Rn	110 Rx	1	disp.32[Rn](Rx)
			disp				11110010	110 Rx	1	disp.32[PC](Rx)
			disp				110 Rn	110 Rx	1	[disp.32[Rn]](Rx)
			disp				11111010	110 Rx	1	[disp.32[PC]](Rx)
			addr				11110011	110 Rx	1	/addr(Rx)
			addr				11111011	110 Rx	1	[/addr](Rx)
disp1				disp2				010 Rn	1	disp1.32[disp2.32[Rn]]
disp1				disp2				11111110	0	disp1.32[disp2.32[PC]]

86-041