

November 1986

**μPD70616
Programmer's
Reference
Manual**

PRELIMINARY INFORMATION

©1986 NEC Electronics Inc./Printed in U.S.A.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics Inc. The information in this document is preliminary and subject to change without notice.

Preface

The ability to integrate hundreds of thousands of transistors on to a single silicon chip represents one of the most important technological advances. This capability has greatly impacted the design of microprocessors and allowed computer architects to design systems in silicon that even a few years ago were unimaginable. As such, the increase in performance of the next generation of 32-bit microprocessors eclipses that of the current generation 16- and 32-bit devices.

Using these semiconductor and computer architecture advances, NEC has been developing the V-Series microprocessor family. A new microprocessor family, the V-Series has been designed from scratch for implementation using high performance, low power CMOS VLSI technology. The V-Series microprocessors are represented in both the high integration and high performance marketplaces. Beginning in 1984 with the introduction of the V20™/V30™ microprocessors and the subsequent introduction in 1985 of the V40™/V50™ microprocessors, NEC has been the world leader in the design and production of high integration CMOS 16-bit microprocessors.

This manual describes the architecture of the first of a family of high performance 32-bit microprocessors, the μPD70616 (V60™). The μPD70616 is a general purpose microprocessor with the features of high-end mainframe and super minicomputers. Featuring a sophisticated 32-bit architecture, the μPD70616 integrates a variety of powerful instructions, data types, addressing modes, virtual memory and a four stage instruction pipeline. Aimed at a wide variety of applications, the μPD70616 will find widespread use in sophisticated office, real-time and engineering systems.

*V20™, V30™, V40™, V50™ and V60™ are trademarks of NEC Corporation

Table of Contents

Section 1	Introduction	1-1
	V-Series	1-1
	μPD70616 Architecture Overview	1-3
	Basic Architecture	1-3
	Virtual Memory Management.....	1-8
	Data Types	1-12
	Instruction Set	1-14
	System Support Features	1-15
	μPD70616 Architecture Implementation	1-17
	Processor Features	1-17
	Pipeline Operation	1-18
	Notational Conventions	1-20
	Numerical Values	1-20
	Data Organization	1-20
	Memory Organization	1-21
	Abbreviations and Special Terminology	1-22
Section 2	Data Types	2-1
	Addressing	2-1
	Byte Addressing	2-1
	Register Addressing	2-3
	Bit Addressing	2-4
	Data Types	2-5
	Signed Integers	2-5
	Unsigned Integers	2-6
	Bit	2-6
	Short Real	2-6
	Long Real	2-7
	Decimal	2-7
	Characters	2-7
	Bit Fields	2-8
	Bit Strings	2-9
	Stacks	2-9

Section 3	Register Set	3-1
Program Register Set		3-3
General Purpose Registers		3-3
Program Counter (PC)		3-3
Program Status Word (PSW)		3-3
Privileged Register Set		3-7
Stack Pointers		3-8
System Base Register (SBR)		3-8
Task Register (TR)		3-9
Task Control Word (TKCW)		3-9
System Control Word (SYCW)		3-11
Processor ID Register (PIR)		3-12
Area Table Registers		3-12
Area Table Base Registers (ATBR)		3-13
Area Table Length Registers (ATLR)		3-13
Address Trap Registers		3-14
Trap Mode Register (TRMOD)		3-14
Address Trap Registers (ADTR)		3-15
Address Trap Mask Registers (ADTMR)		3-15
Program Status Word 2 (PSW2)		3-15
Section 4	Address Spaces	4-1
Introduction		4-1
Virtual Address Space		4-2
Sections		4-3
Areas		4-4
Pages		4-5
Address Space Notes		4-6
Section Notes		4-6
Area Notes		4-7
Protection		4-9
Execution Levels		4-9
Area Protections		4-9
Page Protections		4-9
Memory Address Space		4-10
I/O Address Space		4-10
I/O Space Access		4-10
Virtual Address Space Mapping		4-10
Multiple Virtual Address Spaces		4-11
Address Translation		4-13
Area Table Register Pair		4-14

Area Tables	4-14
Area Table Entries (ATE)	4-14
Page Tables	4-15
Page Table Entries (PTE)	4-16
Section 5 Task Management	5-1
Context Switching	5-1
Instruction Set Support	5-3
Section 6 Instruction Formats and Addressing Modes	6-1
Instruction Formats	6-1
Format I	6-2
Format II	6-3
Format III	6-4
Format IV	6-5
Format V	6-6
Format VI	6-7
Format VII	6-8
Addressing Modes	6-11
Calculation of Bit Addresses	6-12
Addressing Mode Encoding	6-13
μPD70616 Addressing Modes	6-15
Register	6-16
Register Indirect	6-17
Register Indirect Indexed	6-18
Autoincrement	6-19
Autodecrement	6-20
Displacement	6-21
Displacement Indexed	6-22
PC Displacement	6-23
PC Displacement Indexed	6-24
Displacement Indirect	6-25
Displacement Indirect Indexed	6-26
PC Displacement Indirect	6-27
PC Displacement Indirect Indexed	6-28
Double Displacement	6-29
PC Double Displacement	6-30
Direct Address	6-31
Direct Address Indexed	6-32
Direct Address Deferred	6-33
Direct Address Deferred Indexed	6-34

Immediate	6-35
Immediate Quick	6-36
Addressing Mode Restrictions	6-37
Section 7 μPD70616 Instruction Set	
Section 8 Interrupts and Exceptions	8-1
System Base Table	8-1
Interrupt and Exception Processing	8-3
Interrupts	8-4
Exceptions	8-4
μPD70616 Exception Processing	8-5
Serious System Faults	8-5
System Faults	8-5
Stack Invalid Exceptions	8-5
Memory Management Exceptions	8-6
Instruction Exceptions	8-7
Arithmetic Exceptions	8-8
Software Debug Exceptions	8-9
Change Execution Level Exceptions	8-11
Asynchronous Traps	8-11
Emulation Mode Exceptions	8-12
Software Traps	8-13
Interrupt/Exception Stack Formats	8-14
Exception Codes	8-19
Reset	8-20
Interrupt/Exception Nesting	8-21
Interrupt/Exception Stacks	8-27
Section 9 Software Debug Support	9-1
Instruction Trace	9-2
Instruction Trace Control	9-2
Instruction Trace Operation	9-2
Instruction Trace Pending	9-2
UPDPSW.W Instruction	9-3
Instruction Trace Note	9-3
Breakpoint Traps	9-4

Address Traps	9-5
Address Trap Operation	9-5
Address Trap Registers	9-5
Address Trap Setup	9-6
Address Trap Generation	9-7
Virtual/Physical Mode Address Traps	9-7
Address Trap Stack Contents	9-7
Address Trap Notes	9-8
Section 10 V20/V30 Emulation Mode	10-1
Virtual and Physical Address Modes	10-1
Emulation Mode	10-1
Program Status Word (PSW2)	10-2
Program Counter (PC)	10-4
I/O Emulation Option	10-4
Register Allocation	10-4
Emulation Mode Instruction Set	10-6
Instruction Set Summary	10-6
Mode Transitions	10-8
Native Mode to Emulation Mode	10-8
Emulation Mode to Native Mode	10-9
Memory Address Space	10-9
Address Generation	10-9
I/O Address Space	10-10
Emulation Mode Notes	10-10
Section 11 Functional Redundancy Monitor (FRM)	11-1
Fault Tolerant System Configuration	11-1
Functional Redundancy Monitor	11-2
Bus Freeze Interrupt	11-3
FRM Applications	11-4
Bibliography	11-6
Appendix	
Appendix A Instruction Set Summary	A-1
Appendix B Instruction Formats	B-1
Appendix C Addressing Mode Encodings	C-1

Section 1 Introduction

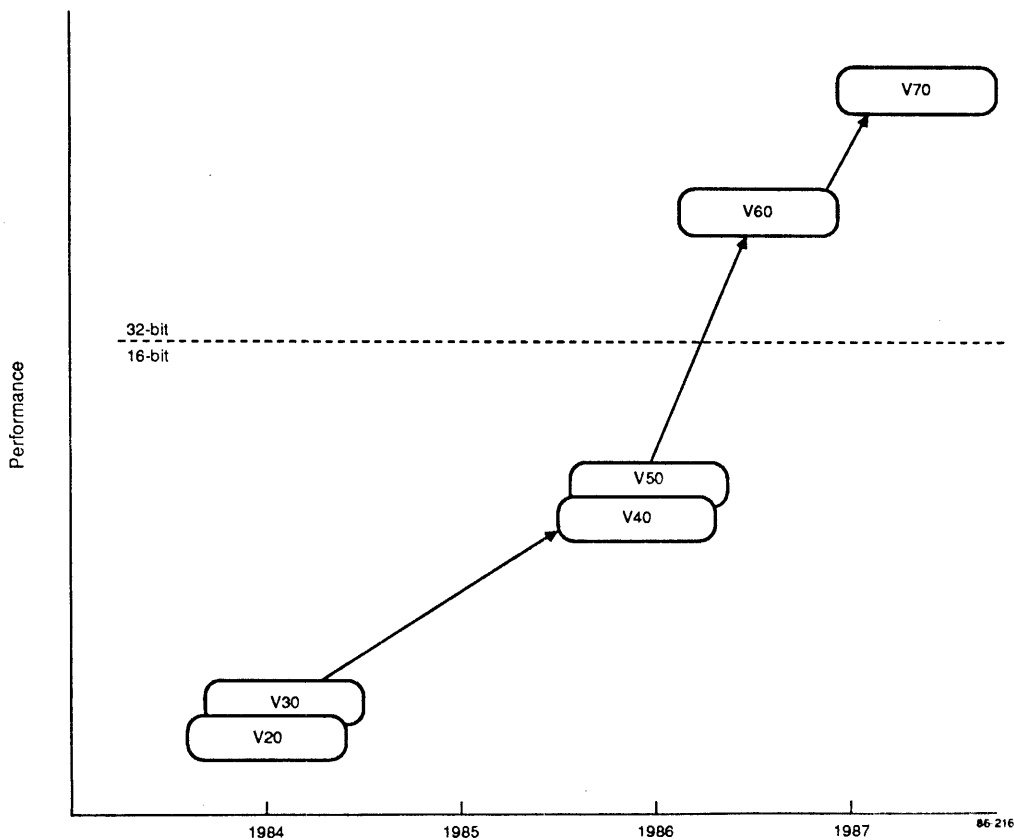
The μPD70616 (V60) is the first member of a family of high performance CMOS 32-bit microprocessors. This section introduces the major features and capabilities of the μPD70616.

V-Series

The V-Series is a set of original, high performance CMOS microprocessors. The V-Series consists of two distinct families, a family of high integration 16-bit microprocessors and a family of high performance 32-bit microprocessors.

The high integration family consists of the μPD70108/116 (V20/V30) and the μPD70208/216 (V40/V50) microprocessors. Using internal 16-bit data paths, these processors have been designed to improve system performance and reduced power consumption while maintaining software compatibility with the previous generation of 16-bit microprocessors.

Figure 1-1. V-Series Microprocessors



The high performance family of 32-bit V-Series microprocessors extend the performance capabilities of the 16-bit V-Series microprocessors. Beginning with the μPD70616 microprocessor, these VLSI devices have been designed from scratch to take advantage of the ability to integrate nearly 400,000 transistors on a single die. The four main objectives which drove the design of the μPD70616 architecture were:

- general purpose architecture
- application hardware support
- expandable
- high performance

These objectives were established in anticipation of the severe performance requirements that will be placed on 32-bit microprocessors by the next generation of applications.

Currently, microprocessors can be found in a wide variety of applications. Because the range of applications is continuing to grow larger, it is becoming increasingly difficult for VLSI designers to predict what will be the requirements of the next generation of machines. This suggests that the design of a microprocessor must take into account flexibility, the ability to mold a microprocessor architecture into many different applications.

Support for high level languages is another design objective given the fact that software is increasingly written using high level languages. Operating systems will also need assistance from the architecture as the complexity and sophistication of both application and system software increases. Virtual memory is a necessary requirement as program complexity grows. Without the protection facilities of a virtual memory system, the development, debug and verification of a large software system would become nearly impossible.

For a variety of reasons, the next generation of applications will standardize on 32-bit architectures that can fit into each market segment. In office automation, it is the ability to manipulate text and numbers that sets one microprocessor over another. In the field of CAD/CAM, the ability to perform high speed numerical calculations dominates the other uses for the systems. In artificial intelligence applications, management of large virtual address spaces and pointers must be efficiently implemented.

At the same time, new applications must be accommodated by the architecture. The conventional architectures which are unable to adapt to an ever changing design environment will be quickly left behind. Microprocessors which do not implement a general purpose architecture, orthogonality or application support are doomed to failure. The μPD70616 microprocessor has been designed with these goals in mind. Unlike other 32-bit processors which are based on obsolete 8- or 16-bit architectures, the μPD70616 has introduced a new architecture rather than extending an architecture that is unsuited for VLSI implementation.

μPD70616 Architecture Overview

Generally speaking, the architecture of a computer refers to the structure and resources available to the programmer and more specifically to the instruction set, register set and address spaces. Thus the μPD70616 architecture refers to the μPD70616 microprocessor facilities and resources as viewed by the programmer. When the μPD70616 architecture is realized in silicon, it becomes an implementation of the architecture. Because of various economic and technical restrictions, the complete architecture may not be fully realized until some later date.

Basic Architecture

The basic μPD70616 architecture is characterized by a general purpose register architecture based on thirty-two 32-bit general purpose registers. The large instruction set is designed to be orthogonal and make full utilization for the register resources. The large number of general purpose registers enables compilers to efficiently allocate registers to improve performance.

The μPD70616 has four execution levels to allow hardware to distinguish between programs with different levels of permissions. Level 0 has complete control of the hardware and is said to be privileged. The other three execution levels are non-privileged and the capabilities of programs at these execution levels is restricted.

Figure 1–2 shows the set of μPD70616 registers, each being a full 32-bits in length. The register set is divided into two parts, the program register set and the privileged register set. Use of the program register set is unrestricted but system level permissions are required to access the privileged register set.

- Program Register Set

The program register set consists of the thirty-two general purpose registers, the PC (program counter) and PSW (program status word). The general purpose registers are each identified by a register ID ranging from 0 to 31 and can be used as temporary storage, accumulators, stacks, base and index registers without restriction. General purpose registers can also be used with the floating point data type making the register set truly versatile.

Of the general purpose registers, three are implicitly selected by instructions and have alternate names. R31 is the SP (stack pointer) and points to the TOS (top of stack) for the current task context. R30 is the FP (frame pointer) and points to the frame activation record of the current procedure context. R29 is called the AP (argument pointer) and points to the list of arguments for the current procedure.

The PC is a 32-bit register that contains the address of the first byte of the current instruction.

The PSW is a 32-bit register which is shared by both the program and privileged register sets. The lower halfword (16-bits) of the PSW contains the condition codes for integer and floating point operations. The upper halfword of the PSW contains system information such as execution level and controls for the software debug and maskable interrupts. The PSW is shown in Figure 1–3.

- Privileged Register Set

Privileged registers are accessible by programs executing at execution level 0 and are used primarily by the operating system to manage the system. The privileged registers are broken down into four functional groups:

- stack pointers
- memory management registers
- system management registers
- software debug registers

Figure 1–2. μPD70616 Register Set

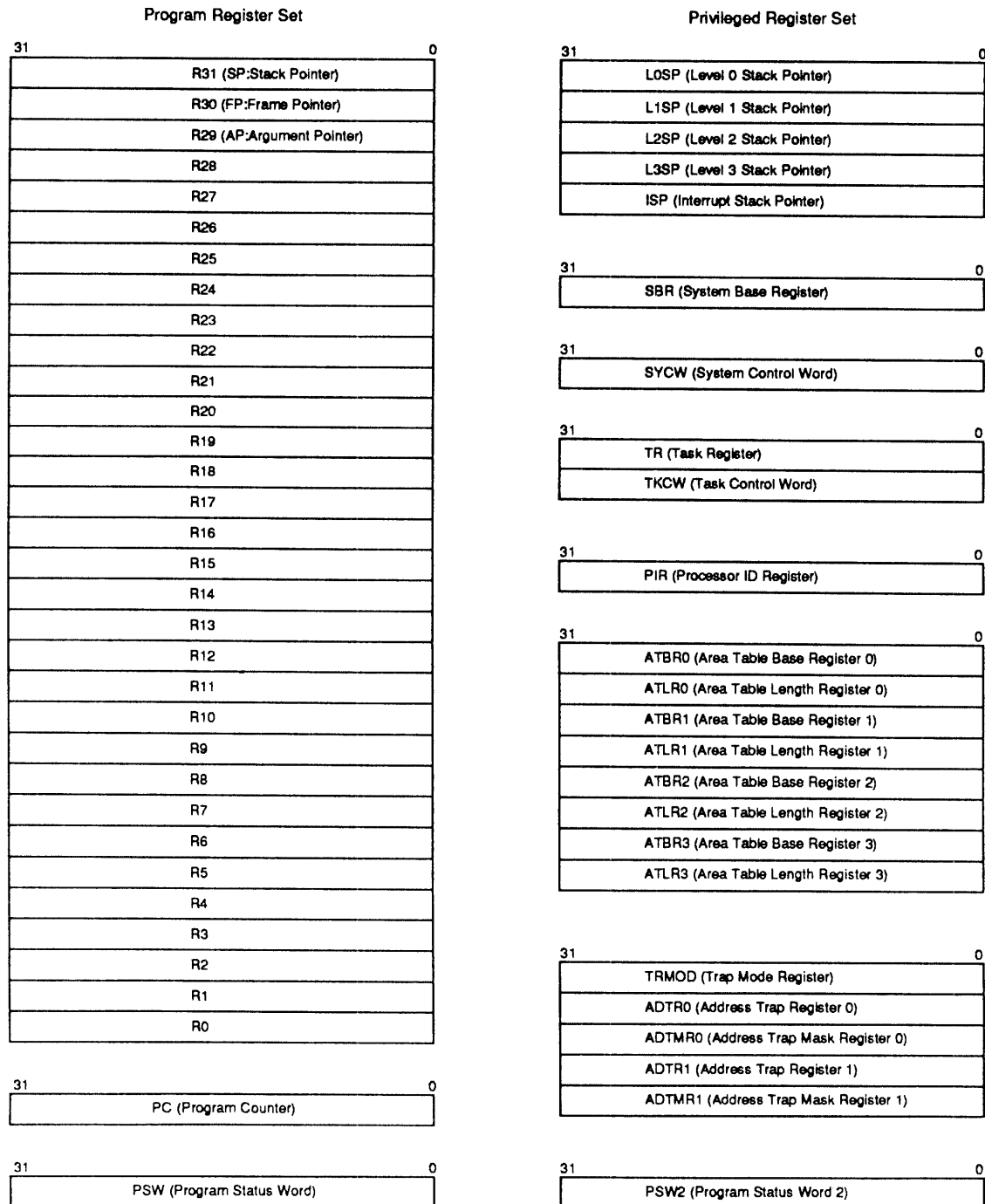
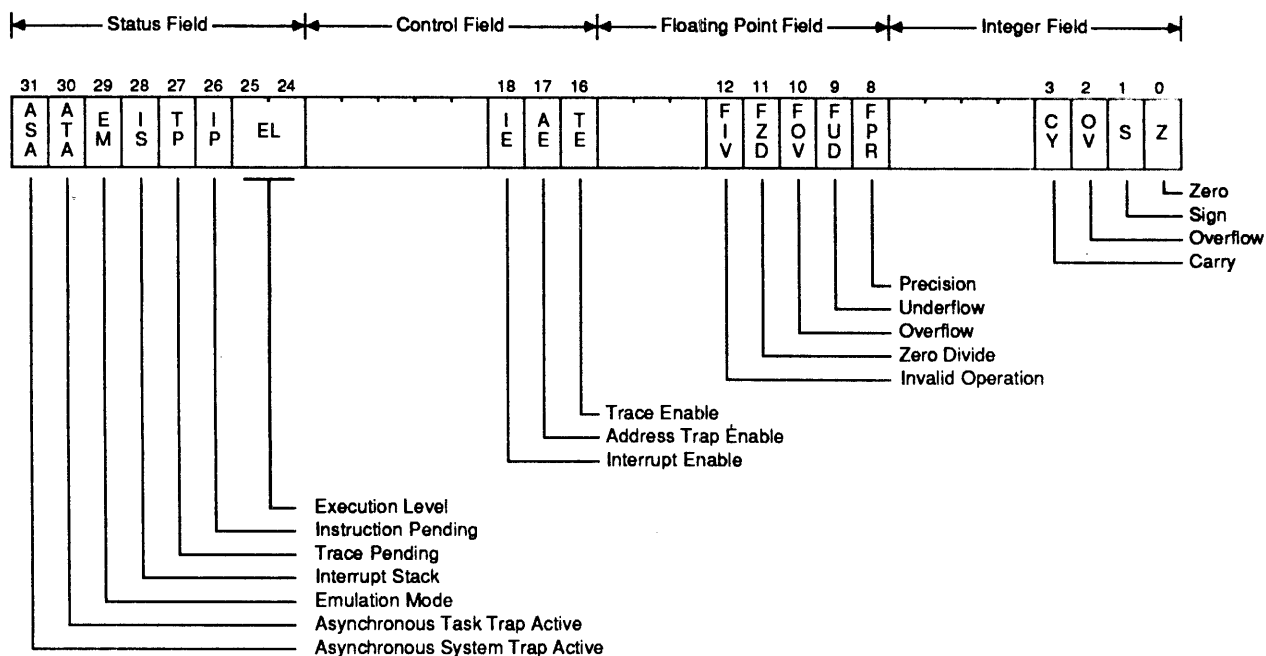


Figure 1-3. PSW Format



86-217

Each of the execution levels has its own stack pointer with a separate interrupt stack pointer for a combined total of five stack pointers. The SP register is actually one of these registers depending on the current execution level. Each time the execution level changes or an interrupt occurs, the μPD70616 will automatically save the current stack pointer and switch to the new stack pointer.

The memory management registers are used to keep the base address and length information for the memory resident address translation tables. The memory management register set consists of four pairs of area table base and length registers.

The system management registers are used to control the operation of the system. The task register points to the task control block of the current task context while the task control word defines the operating environment for the task. The system-wide operating environment for virtual address spaces and asynchronous traps is contained in the system control word. The system base register holds the base address of the system interrupt exception vectors. PSW2 is the PSW used by emulation mode programs.

The five software debug registers are used for controlling address traps. One register is used as an enable for the traps and the other four consist of two pairs of address and address mask registers.

• Instruction Formats

A μPD70616 instruction can range in length from 1 to 22 bytes depending on the type instruction, the number of operands and the operand addressing modes. The μPD70616 is a two address machine with the addressing mode assignment specified independently for each operand. Using independent operand addressing modes, a single ADD opcode can be used to implement register-register, memory-register and memory-memory operations.

- Addressing Modes

A large number of powerful addressing modes is another distinguishing characteristic of the μPD70616 architecture. The μPD70616 has 21 addressing modes for byte addressable data and 18 addressing modes for bit addressable data.

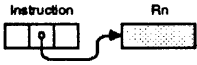

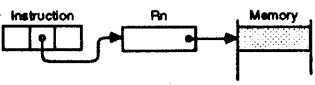
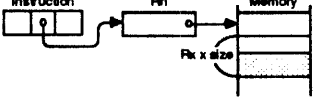
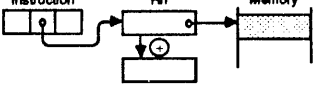
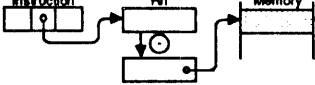
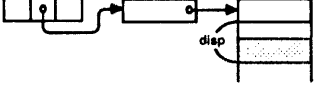
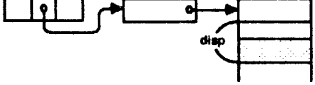
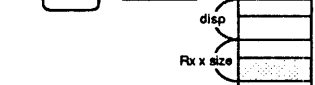
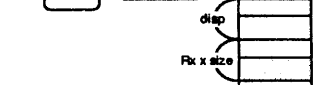
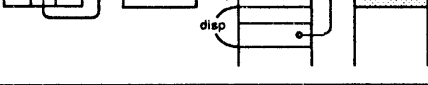

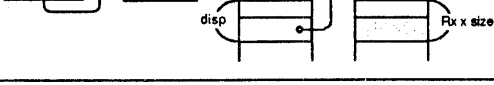
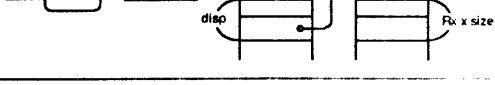
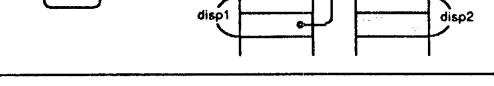
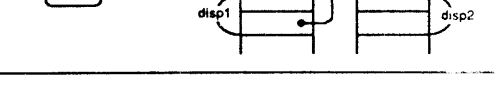

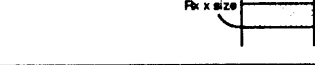


In addition to the standard addressing modes, other enhancements have been made. Three different size displacements can be specified allowing an assembler to optimize the size of the displacement field. Indirect memory addressing supports the use of pointers and any of the general purpose registers can serve as a base or scaled index register. To promote the use of position independent PC relative addressing, the PC can also be specified as a base register. Table 1–1 and Figure 1–4 depict the various addressing modes available.

Table 1–1. μPD70616 Addressing Modes

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

86-218

Figure 1-4. Addressing Mode Operations

Addressing Mode	Operand Addressing	Addressing Mode	Operand Addressing
Rn		#value #value (0-15)	
[Rn]		[Rn](Rx)	
[Rn+]		[-Rn]	
disp[Rn]		disp[PC]	
disp[Rn](Rx)		disp[PC](Rx)	
[disp[Rn]]		[disp[PC]]	
[disp[Rn]](Rx)		[disp[PC]](Rx)	
disp1[disp2[Rn]]		disp1[disp2[PC]]	
/addr		/addr(Rx)	
[/addr]		[/addr](Rx)	

86 219

Virtual Memory Management

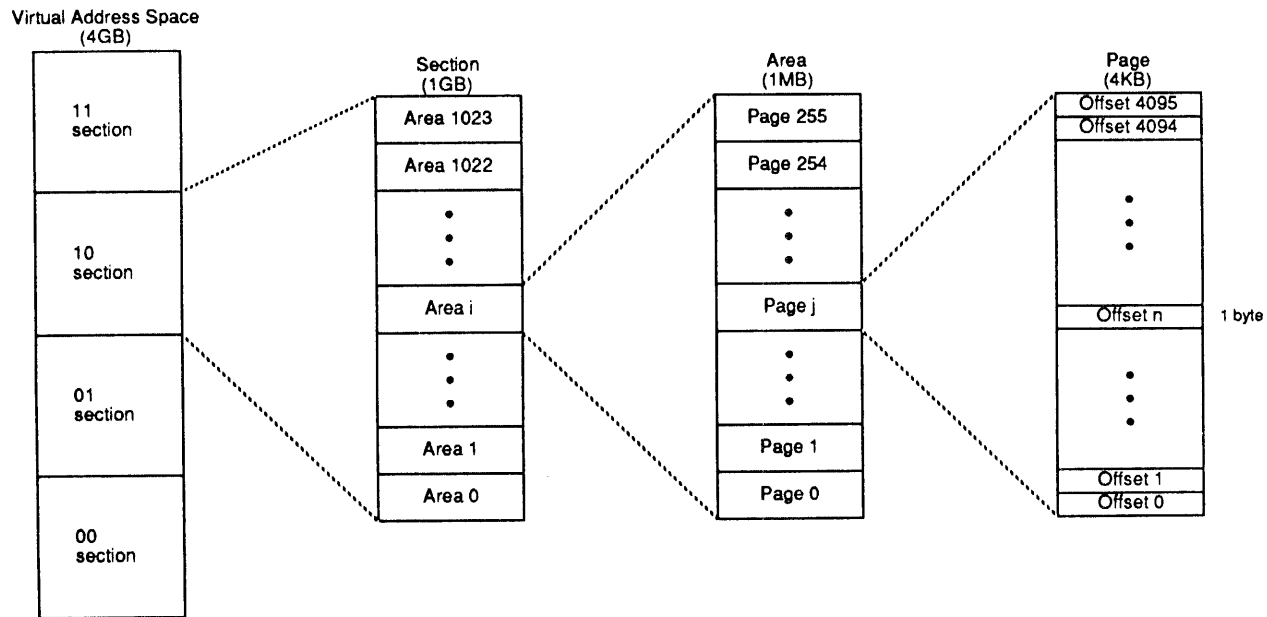
The μPD70616 incorporates an on-chip MMU (memory management unit) which maps each 4GB virtual address space into a 16MB physical address space. The MMU permits each task to have an independent virtual address space or to share a subset with one or more other tasks. Protection, preventing tasks with insufficient privilege from accessing code or data, is also enforced by the MMU.

The linear 4GB virtual address space is divided into 4KB pages for the purpose of demand paging. Demand paging permits portions of the virtual address to be swapped out to low cost secondary storage. This allows programs to be written without regard to the amount of physical memory because the operating system and MMU provides an illusion of a full 4GB physical memory.

• Address Space Structure

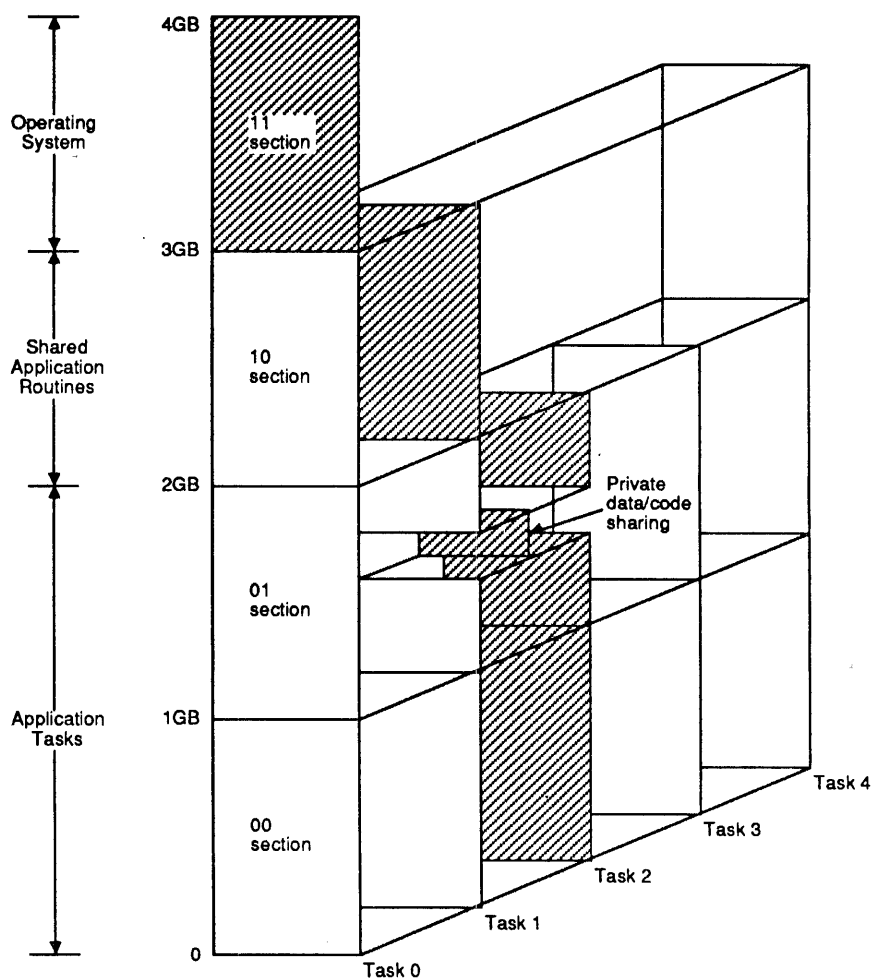
The size of a virtual address space can range as high as 4GB. An operating system is capable of managing multiple virtual address spaces where each space is independent or shared as shown in Figure 1–6. Address space sharing can be implemented at different levels depending on the individual requirements. Each virtual address space is split into four 1GB sections. A section is the first unit of shared virtual address space. For example, sections 10 and 11 can be common to all tasks and contain the system utility and operating system programs while section 00 and 01 are unique to each task. Each section is further divided into 1024 areas. An area is a 1MB region and can be shared between two tasks and used for inter-task communication. An area is broken down into 256 pages, each 4KB in size. The page is the basic unit of memory management and the virtual memory facility is implemented by swapping pages.

Figure 1–5. μPD70616 Address Space



86-220

Figure 1-6. Multiple Virtual Address Spaces



86-221

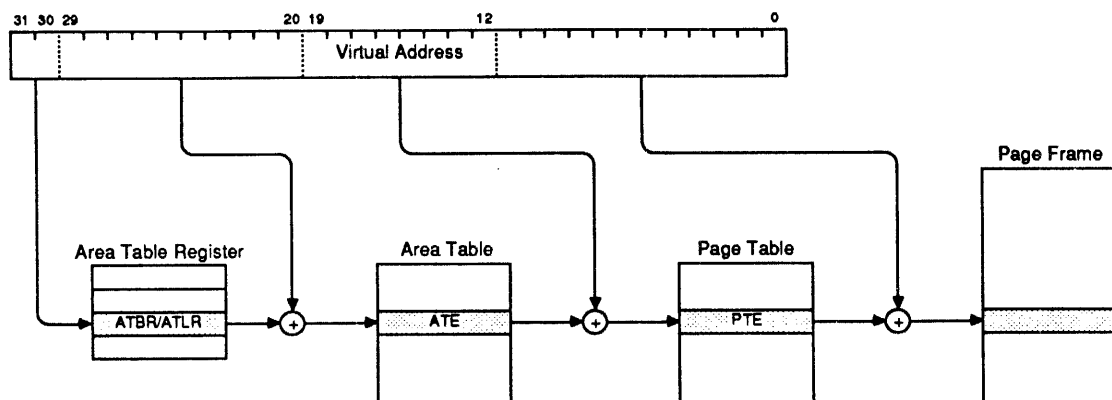
- Protection Mechanisms

Protection mechanisms are implemented at the both the area and page levels. Area level protections compare the current execution level with the execution level required for the particular access. Page level protections are used to mark the page as readable, writable or executable. An access is permitted only when both protection criteria have been satisfied, otherwise a memory management exception will occur.

- Address Translation

When virtual mode is enabled, virtual addresses must be translated into physical addresses before the access is performed. Address translation involves an area table register pair, an area table and a page table. Refer to Figure 1–7 for an example of address translation.

Figure 1–7. Address Translation



86-222

The 32-bit virtual address is split into fields for the purpose of address translation. First, the upper two bits are used to identify the section by selecting one of the four area table registers pairs. The area table register pair contains a the base address and length of the memory resident area table to be used in the next step of the translation.

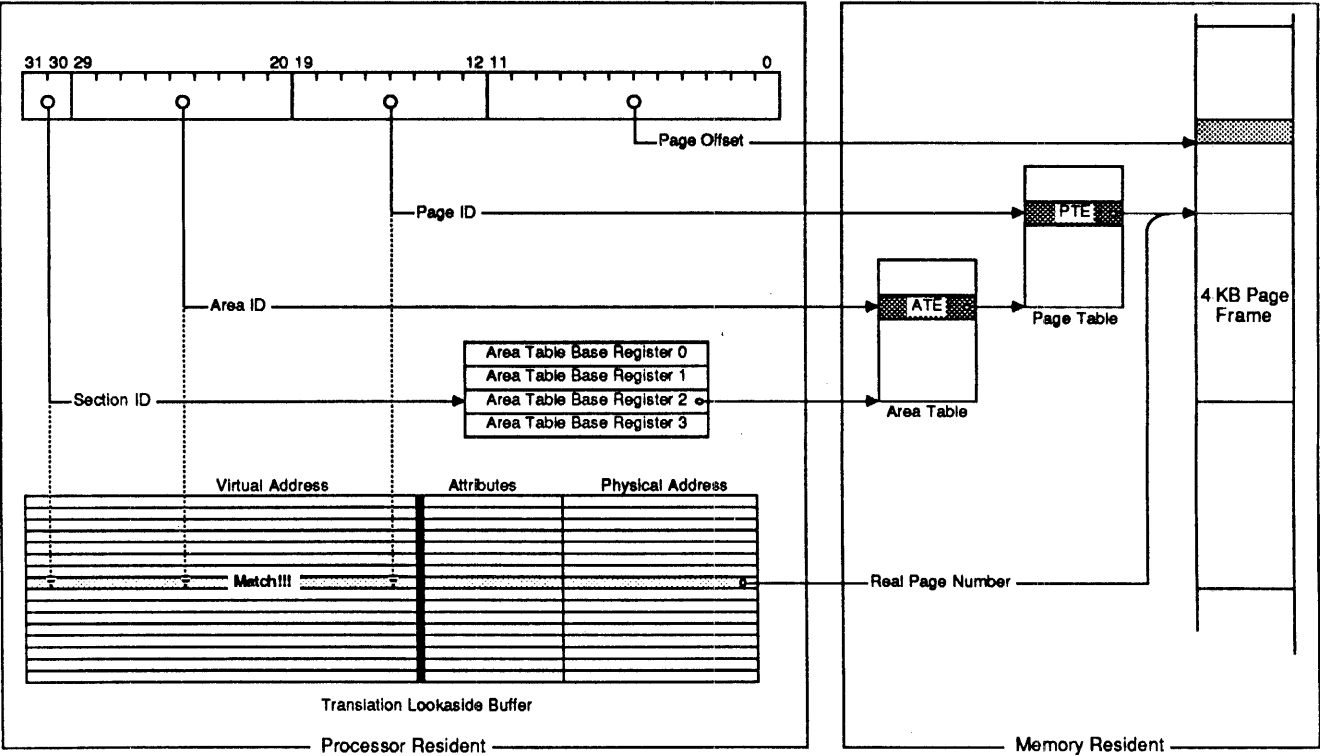
Using the base address from the previous step, bits 20:29 of the virtual address are used as the offset into the area table to select the ATE (area table entry). The selected ATE contains the base address of the page table to be used in the second level of translation and the permissions required to access this area. If the access permissions are not met, the translation is aborted and an exception will occur.

Next, bits 12:19 of the virtual address are used as an offset in the page table to select a PTE (page table entry). The PTE contains the physical base address of the page, whether it is physical present and the page level access permissions.

If the page level protections are met, the access can occur and low order 12 bits of the virtual address are concatenated with the page base address obtained from the PTE to form the physical memory address.

Address translation and the associated table lookups are an inherently slow process and a hardware assist is required. Accesses to programs and data are generally sequential and localized. The μPD70616 can take advantage of this and cache the last 16 address translations on-chip in a high speed TLB (translation look-aside buffer). If the section, area and page ID fields match an entry in the TLB and the permissions are satisfied, the address translation will occur immediately and without any performance penalty.

Figure 1-8. Virtual → Physical Address Translation



86-203

Data Types

The μPD70616 supports a wide range of data types and instructions to operate on each data type. These data types include not only the widely used integer and character string data types but also bit, bit field and bit string data types. The μPD70616 also supports IEEE standard floating point arithmetic as part of the basic instruction set.

- **Signed Integer**

Signed integers are represented in two's complement format in byte (8-bit), halfword (16-bit), word (32-bit) and doubleword (64-bit) sizes. A wide range of data transfer and arithmetic operations on signed integers are supported.

- **Unsigned Integer**

Unsigned integers are available in byte (8-bit), halfword (16-bit), word (32-bit) and doubleword (64-bit) sizes. A wide range of data transfer, arithmetic and logical operations on unsigned integers are supported.

- **Floating Point**

The μPD70616 floating point data types and operations conform to the IEEE 754 standard. Floating point data can be represented in either the short real (32-bit) or long real (64-bit) formats. Like integers, floating point data can reside in any of the general purpose registers. Instruction set support for floating point data types includes data transfer, arithmetic and conversion operations.

- **Pointer**

In the μPD70616 a pointer is represented as an unsigned 32-bit integer. Instruction set support for the pointer data type allows calculation of the effective address of operands for parameter passing.

- **Decimal**

The decimal data type is used for calculations in BCD format. Addition, subtraction and conversion instructions are available for the decimal data type.

- **Bit**

The bit data type refers to a single bit within a word. Bits are typically used as Boolean variables and can be set, cleared, complemented and tested.

- **Bit Field**

Bit fields are variable length data structures that are a subset of the integer data types. Bit fields range from 0 to 32 bits in length and are used to pack signed and unsigned integers. Instructions are available to insert, extract and compare bit fields.

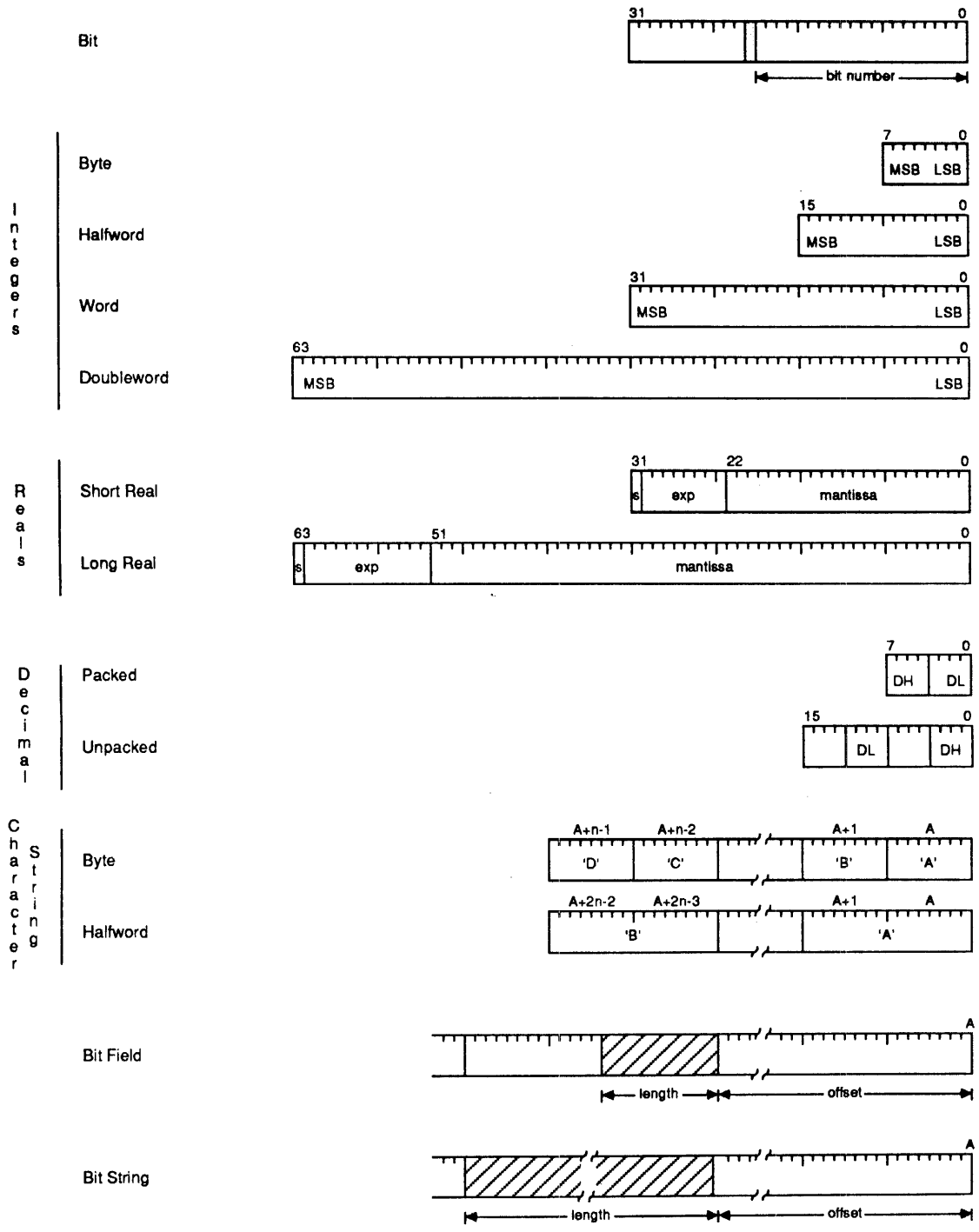
- **Bit String**

Bit strings are variable length logical data structures ranging from 0 to 4 gigabits in length. Bit strings begin and end on any arbitrary bit boundary. Any logical operation can be performed on bit strings as well as searching bit strings for a 0 or 1 bit.

- **Character String**

Character strings are used to efficiently manipulate text using either byte or halfword characters. Operations on character data include data transfer, comparison and searching.

Figure 1-9. Data Type Formats



86-223

Instruction Set

Besides the comprehensive instruction set for each of the data types, the μPD70616 instruction set contains support for high level languages, context switching and other applications. Instructions which manipulate system hardware or data structures are privileged and require the processor to be at execution level 0.

- **Control Transfer**

Control transfer instruction set support includes conditional and unconditional branches, looping and subroutine call/return instructions.

- **Procedure Calling**

Procedure calling is an important requirement for the efficient execution of high level languages. The μPD70616 supports procedure calls and returns as well as instructions for the management of the local stack frames. Figure 1–10 is an example of a typical procedure call and return.

- **Context Switching**

Rapid switching of task contexts is a requirement of any multi-tasking operating system. The μPD70616 instruction set supports this requirement by loading and storing task contexts with a single instruction.

- **Virtual Memory Management**

Virtual memory requires the maintenance of translation tables and the TLB (translation look-aside buffers). A number of instructions are dedicated to read and update table entries, translate a virtual address to a physical address and control the operation of the memory management unit.

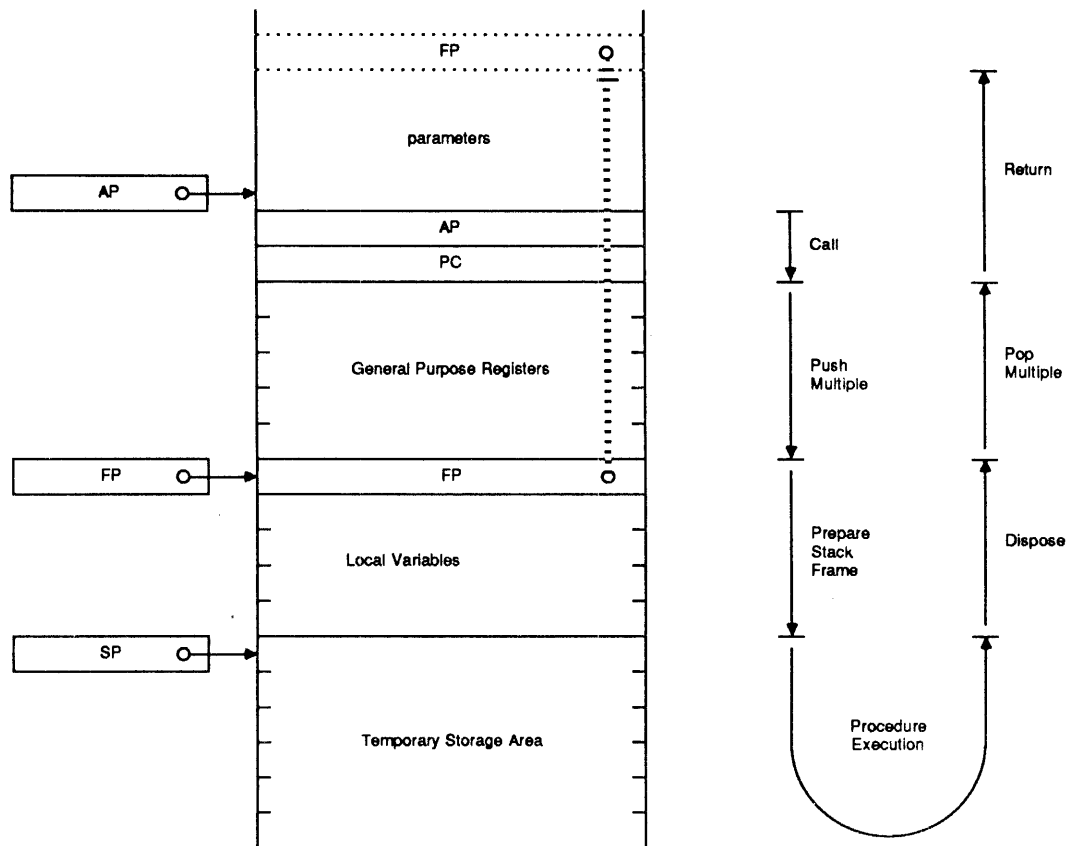
- **Input/Output**

Input/output instructions are used to communicate with external peripheral devices in the I/O address space. Peripherals can be accessed either using the standard I/O instructions or via the virtual I/O facility by any instruction that references memory.

- **Multiprocessing**

In order to support the efficient implementation of multiprocessor systems, the instruction set primitives for test and set and compare and exchange operations are provided.

Figure 1-10. Procedure Call/Return



86 224

System Support Features

System support features are those facilities that assist in the development of both system and application software. The μPD70616 provides a number of aids for the implementation of operating systems and the debug of programs.

- **Asynchronous Traps**

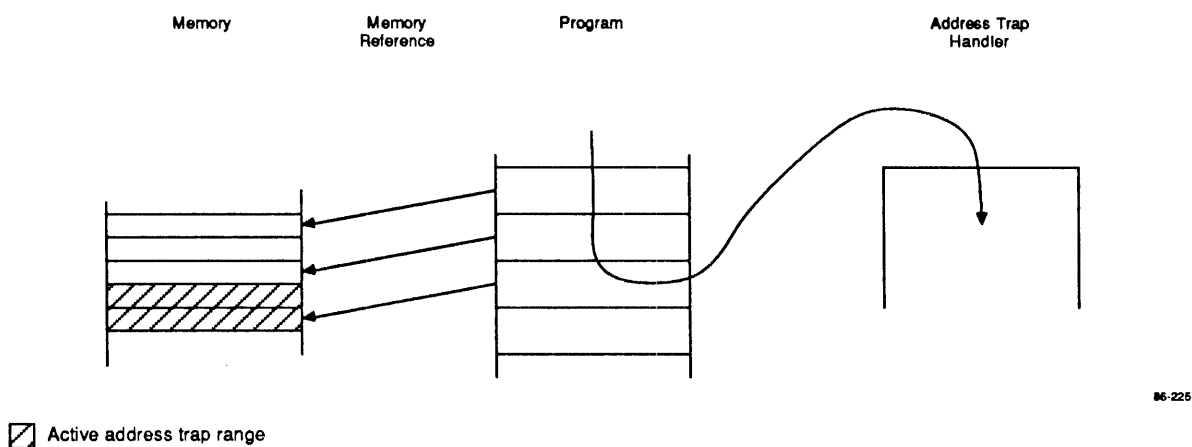
Asynchronous traps are designed to simplify the implementation of multi-tasking operating systems. Many operating system services are interrupt driven and a dilemma is presented by how to inform the operating system of asynchronous events in a well defined and consistent manner. Asynchronous traps provide this consistent interface between operating system routines.

This communication between operating system routines is called an AST (Asynchronous System Trap). The ATT (Asynchronous Task Trap) is a similar facility used by the operating system to inform a task of an event.

- Software Debug Support

As the size and complexity of application and system software grows, so to does the task of debugging the program. The μPD70616 architecture provides three powerful software debug aids designed to minimize the amount of time spent debugging programs. Instruction trace and instruction breakpoints are standard tools used in software debug. In addition, a powerful tool called address traps is also available. Address traps combine a region of addresses with the type access (read/write/execute). Each memory access is checked and a trap will occur whenever the trap criteria is met. The capabilities are further enhanced by providing two independent sets of address trap registers for use in software debug.

Figure 1-11. Address Traps



μPD70616 Architecture Implementation

The μPD70616 is the first realization of the V-Series 32-bit architecture in a high performance CMOS VLSI process.

Processor Features

In addition to the implementation of the 32-bit V-Series architecture, the μPD70616 includes the following additional features:

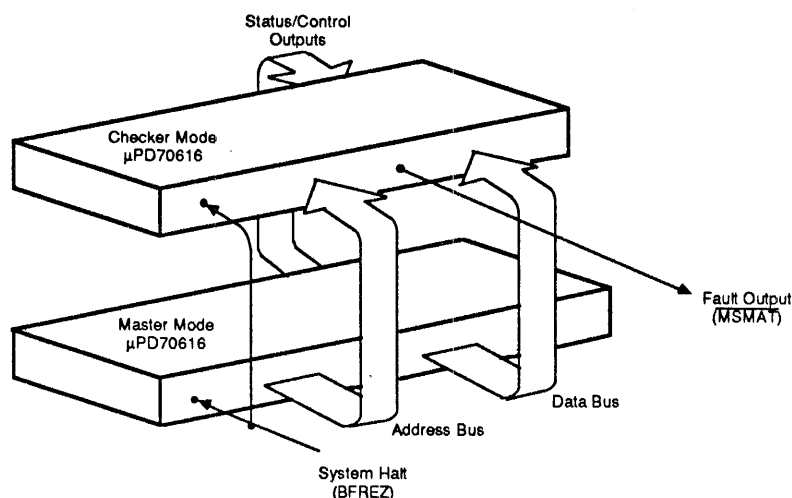
- Emulation Mode

Emulation mode allows software developed for the μPD70108/116 (V20/V30) microprocessors to be executed. Emulation mode is directly compatible with object code and no re-assembly or re-compilation is required. During emulation mode, all of the software debug, memory management and protection mechanisms are fully functional, thus multiple emulation mode programs can co-exist in the same virtual address space without interference.

- Functional Redundancy Monitor (FRM)

A functional redundancy monitor is a technique used to increase the reliability of a system by allowing a redundant processor to check the operation of the master processor. Using FRM techniques, the design of fault tolerant computing systems is simplified.

Figure 1-12. Functional Redundancy Monitor



86-226

Pipeline Operation

The concurrent processing of multiple instructions is made possible by using independent functional units with interlocks. Extensive instruction pipelining is used in the μPD70616 to keep each functional unit busy and maximize the instruction throughput. Each instruction utilizes the following functional blocks during the course of execution:

- PFU (Pre-fetch Unit)

The pre-fetch unit is designed to load the 16 byte instruction queue from external memory during idle bus periods. Since instruction execution is generally a sequential process, the latency to fetch an instruction from memory can be reduced to zero if the instruction is found in the pre-fetch queue.

- IDU (Instruction Decode Unit)

Pre-fetched instructions are fed to the instruction decode unit to be decoded. Each instruction is examined for operands and the addressing mode information is stripped out and sent to the effective address generator to calculate the addresses for processing. The decoded instruction is then queued up for processing by the execution unit.

- EAG (Effective Address Generator)

The effective address generator uses a high speed multi-way adder to quickly compute the virtual address of an operand and sends it to the memory management unit for translation.

- MMU (Memory Management Unit)

In the virtual mode, addresses must be converted to a physical address. Using a 16 entry TLB (translation look-aside buffer) and pipelining the address translation, the effective address translation time is zero. Meanwhile, in parallel with the address translation, the access permission is verified. The output of the MMU is a physical address for the bus control unit.

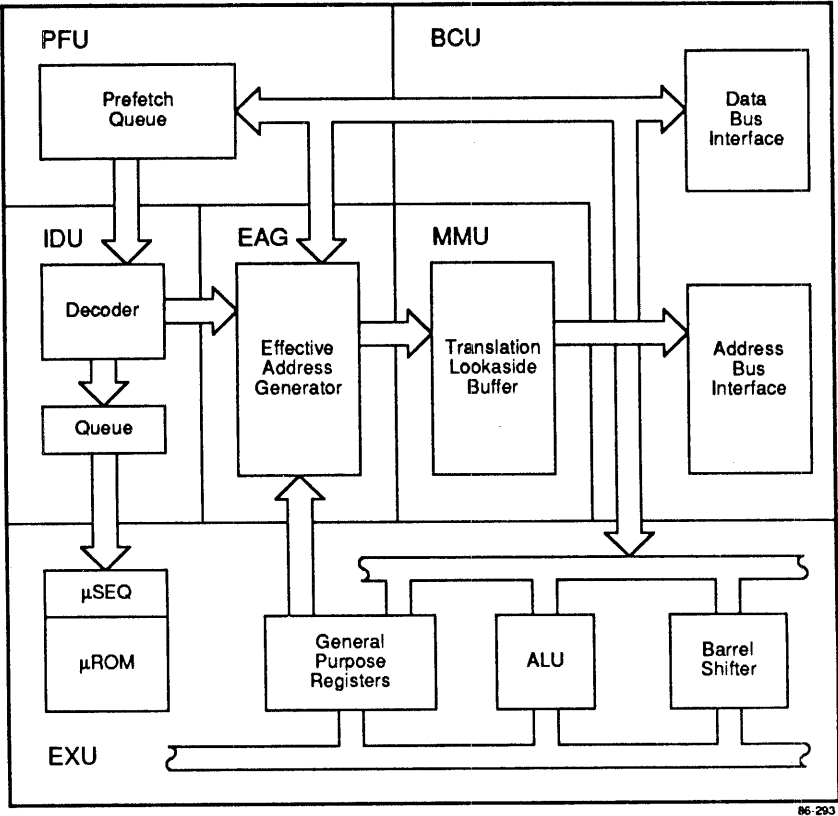
- BCU (Bus Control Unit)

The bus control unit acts the interface for each internal bus requester. Additional logic allows the BCU to re-run faulty bus cycles in the event of an memory ECC error and to use a short cycle bus mode for accessing fast cache memories. The functional redundancy monitor is also implemented in the BCU.

- EXU (Execution Unit)

The actual execution of an instruction occurs in the execution unit. Instruction execution begins when the instruction has been decoded and all operands have been fetched. The EXU is composed of a 32-bit microprogrammable ALU with the thirty-two general purpose registers, and a high speed barrel shifter.

Figure 1-13. μPD70616 Functional Blocks and Pipeline



Notational Conventions

The terminology and abbreviations used in this document are defined for the reader's convenience in this section.

Numerical Values

Numerical values are normally expressed as decimal numbers, but it is sometimes clearer to use other notations. The suffixes 'b' and 'H' will sometimes be used to indicate numbers written in binary (base 2) and in hexadecimal (base 16). This provides three ways of writing numbers. For example, the (decimal) number 118 can be represented as 01110110b or 76H.

The prefixes "kilo", "mega" and "giga" are also widely employed. They correspond to the following values:

Table 1-2. Numeric Prefix Values

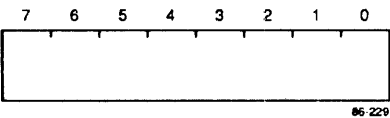
Symbol	Prefix	Value
K	kilo	$1,024 = 2^{10}$
M	mega	$1,048,576 = 2^{20}$
G	giga	$1,073,741,824 = 2^{30}$

Data Organization

The fundamental unit for the addressing of data in memory is the byte. Data that spans multiple bytes are addressed by the address of the least significant byte.

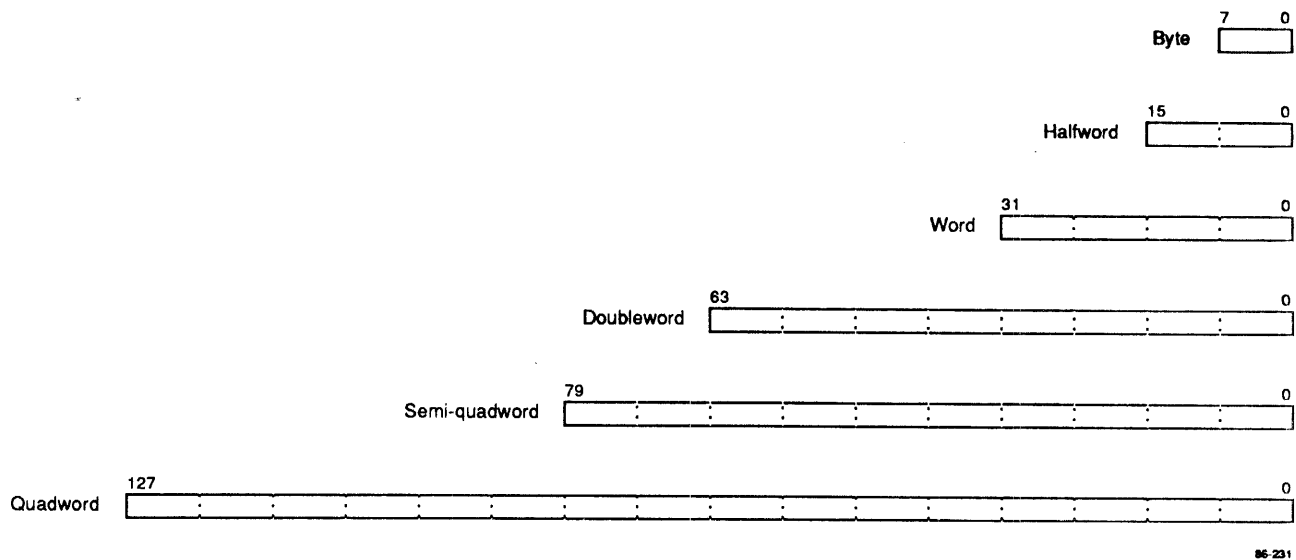
- Byte Representation
A byte consists of eight bits which are numbered from right to left starting with 0. When a byte contains an integer, the MSB (most significant bit) is bit 7 and the LSB (least significant bit) is bit 0.

Figure 1-14. Bit Position within a Byte



- Multiple Byte Representations
When representing larger data types, the ordering of bits and bytes is again from right to left. The rightmost bit is the LSB and the leftmost bit is the MSB. The rightmost byte is the least significant byte and its address is used to access the data.

Figure 1–15. Multiple Byte Representations



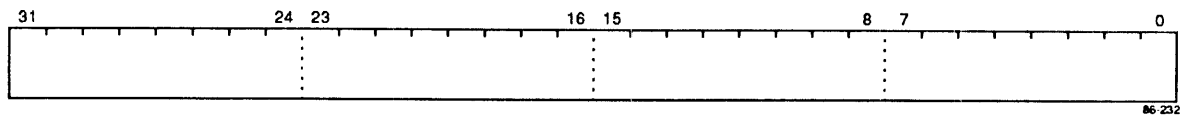
- **Bit Expressions**

Bits are expressed by a bit number. When expressing several consecutive bits within a register or memory location, the bit field expression is used. For example, the bits from bit position 3 to bit position 5 can be identified as bits 3:5.

- **Register Expressions**

Registers are fixed length internal data storage locations. Like memory, registers consist of bits and bytes starting from and numbered from the left. Data stored in a register has the same representation as when that data is stored in memory.

Figure 1–16. Register Representation



Memory Organization

- **Addresses**

Each byte in memory is identified by an address. Starting from address 0, addresses are assigned sequentially up to address 4,294,967,295 ($2^{32} - 1$, or 4 gigabytes). Before address translation, an address is a virtual address while after address translation it is a memory or I/O physical address.

- **Data Organization**

Data is stored in memory in byte units with the address of the lowest byte used to address the data. For example, in Figure 1–18, a word is a four byte data structure and is addressed by the least significant byte of the word.

- **Bit Data**

Data types that use a bit addressing mode require a 35-bit address. A bit address consists of 32 bits used to address the byte while the remaining 3 bits are used to select the bit within the byte. A bit address can be formed by simply appending the bit offset within a byte to the byte address.

- **Data in Memory**

When the address of data is a multiple of the size of the data type in bytes, the data is said to be aligned. Byte data is always aligned and halfword, word, doubleword and quadword data are aligned when they have addresses that are a multiples of 2, 4, 8 and 16 respectively. Semi-quadword data is a special case and is aligned on quadword boundaries.

In some special cases, instructions and data must be aligned. However, generally there are no alignment requirements and only the performance is affected by not aligning data on its boundary.

Abbreviations and Special Terminology

- **μPD70616**

The terms μPD70616 architecture and μPD70616 microprocessor are used throughout this document. The μPD70616 architecture is used to refer to the specification of the 32-bit V-Series microprocessor family. The term μPD70616 microprocessor is used to refer to the silicon implementation of this architecture.

- **UNPREDICTABLE**

The results of an operation designated by the word "UNPREDICTABLE" may vary, depending on the implementation of the processor. Such operations are considered abnormal and illegal, and there is no guarantee that the operation will be handled consistently across all microprocessors.

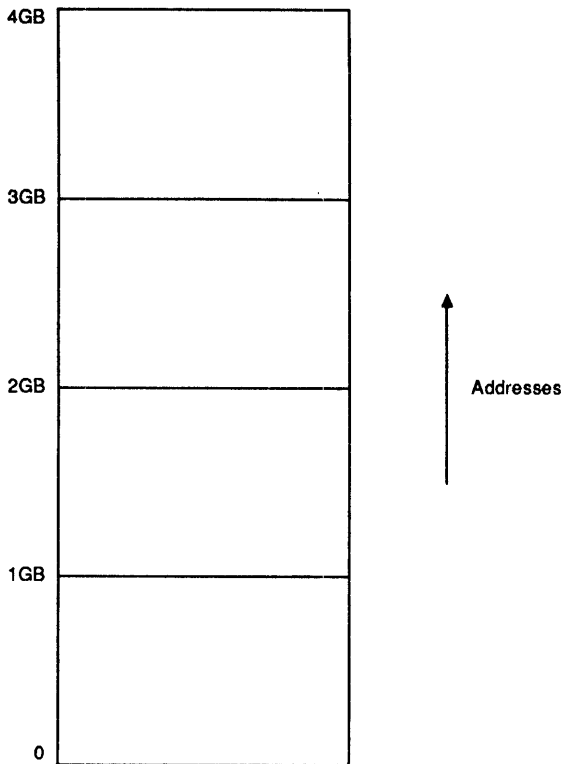
- **MBZ (Must Be Zero)**

MBZ indicates that all the bits in the specified data field must be 0. If software sets a value which is non-zero, the results are UNPREDICTABLE.

- **RFU (Reserved for Future Use)**

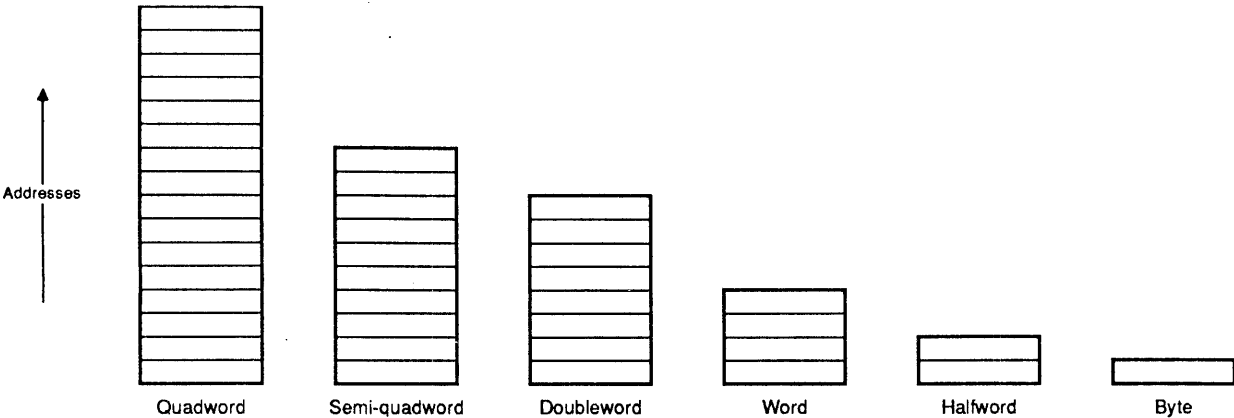
RFU indicates that a data field is reserved for future use by NEC. Fields marked as RFU are implicitly selected as MBZ. Software which uses such fields is not portable and the results of such use are UNPREDICTABLE.

Figure 1-17. Address Space Representation



86 233

Figure 1-18. Byte Addressed Data



86 234

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

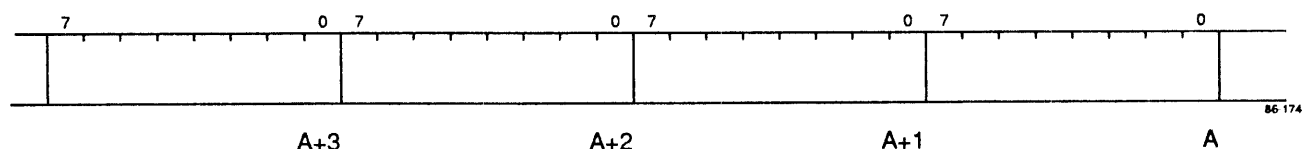
Variable Length Data Types

- Character String
- Bit String
- Bit Field

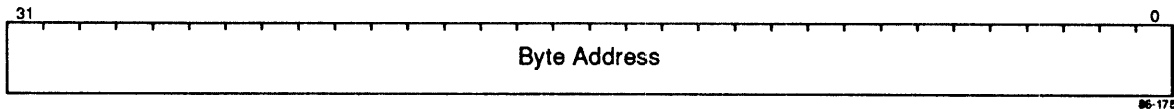
Also important is the organization of the data types. Data type organization describes the layout of a data type in a register or in memory. The μPD70616 architecture is flexible in that both byte and bit addressing modes are available. Byte addressing uses the byte (8-bits) as the basic addressing unit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



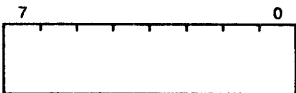
Byte addressing is also used for fixed length data types and for the variable length character string data type. These data types share the characteristic that they are always aligned on a byte boundary, irregardless of the size of the data type. Since byte addresses are 32-bits wide, a byte address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



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

Byte

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

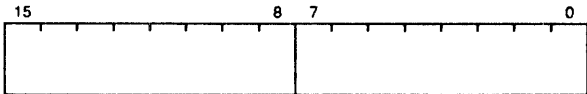


A

86-181

Halfword

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



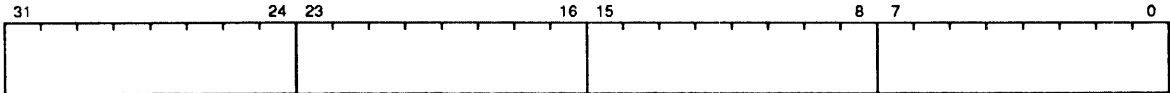
A+1

A

86-182

Word

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



A+3

A+2

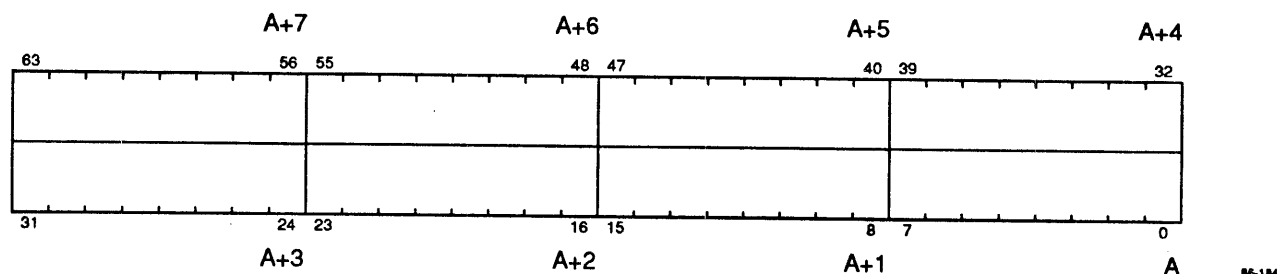
A+1

A

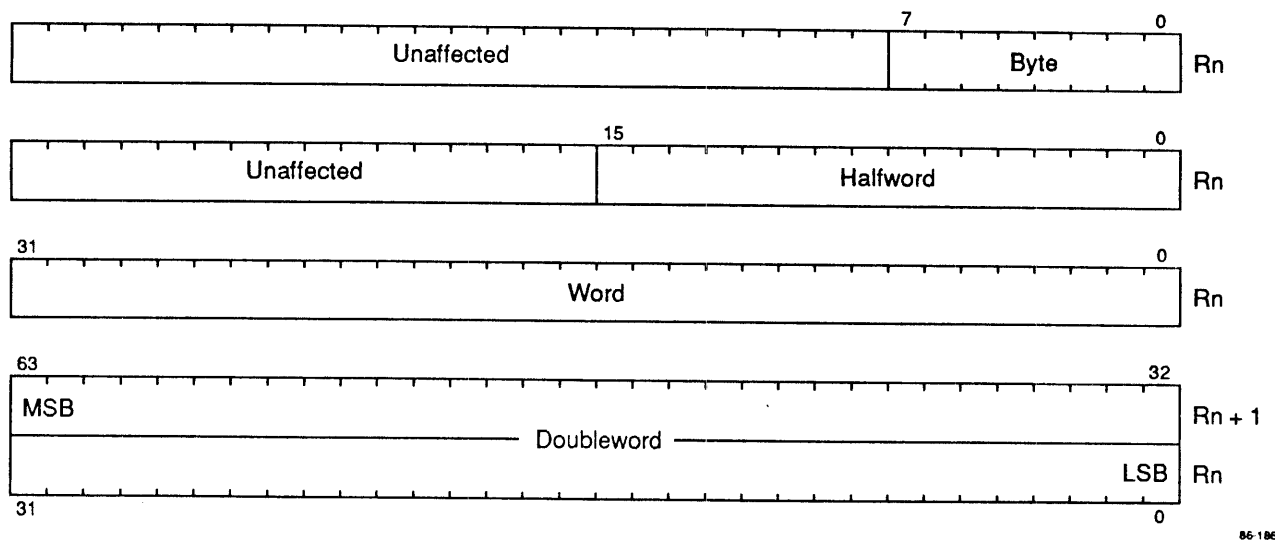
86-183

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:

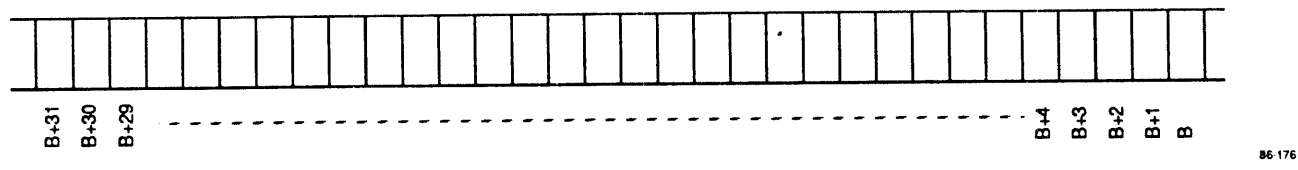


The lengths of the byte and halfword access types are shorter than the register length. These access types are right justified within the register. Only the lower portion of the register corresponding to the access type is significant and the upper portion will remain unaffected.

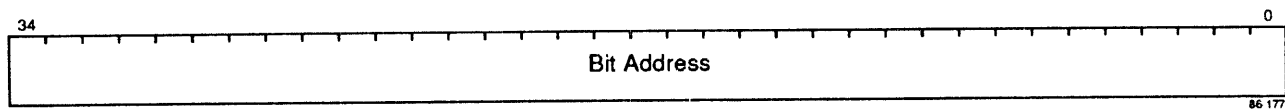
In the case of the doubleword access type, the operand occupies a pair of general purpose registers. The lower numbered register contains the least significant word while the higher numbered register contains the most significant word. However, since R31 cannot be used as the least significant register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

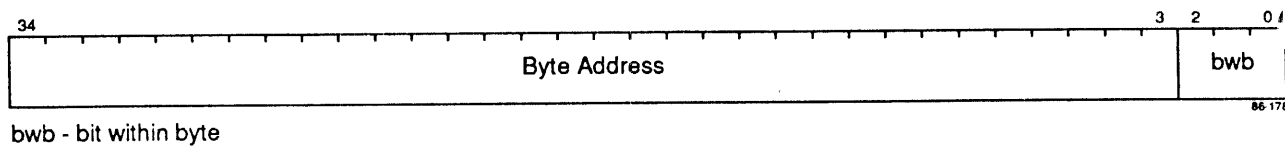
Bit addressing is employed to address data structures that are bit aligned, i.e., are aligned on an arbitrary bit boundary. However, unlike byte addressing which uses the byte as the atomic unit for memory addressing, bit addressing uses the bit as the basic addressing unit. Instead of using a four gigabyte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



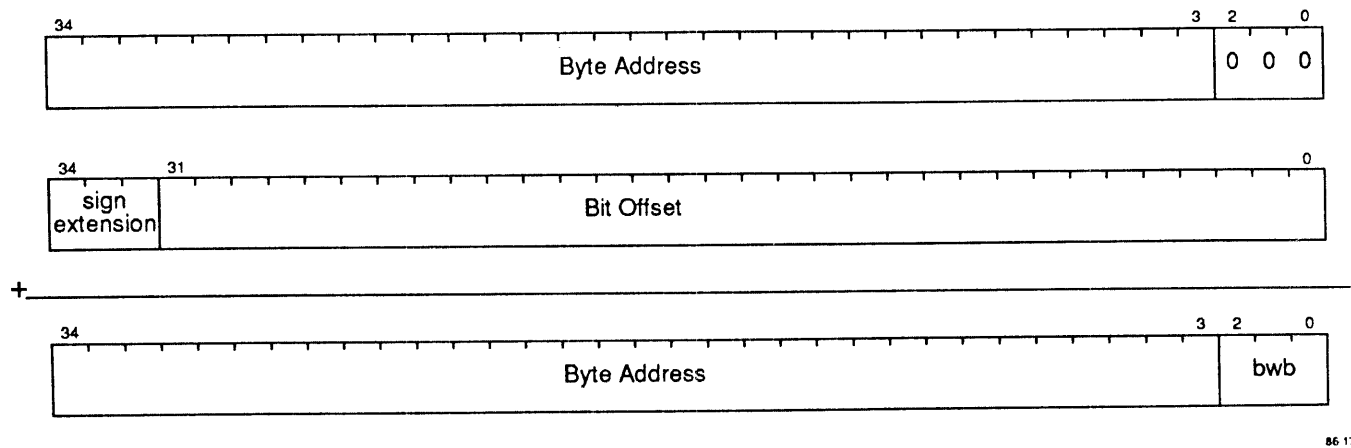
To address an arbitrary bit within a thirty-two gigabit address space requires a 35-bit address since there are 232 bytes, each containing eight bits. A bit address is composed of two separate components, a 32-bit byte base address and a 32-bit bit offset. These components are combined to generate the 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.



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

B6-180

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.

<u>Data Type</u>	<u>Range</u>
Byte	-128 ~ 127
Halfword	-32768 ~ 32767
Word	-2147483648 ~ 2147483647
Doubleword	-9223372036854775808 ~ 9223372036854775807

Unsigned Integers

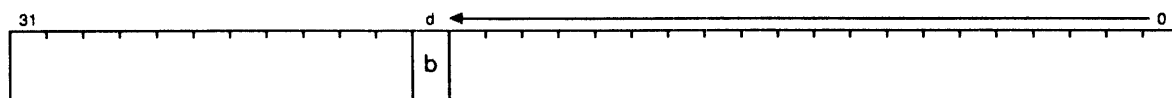
Unsigned integers represent natural numbers (non-negative integers) in binary notation. There are four unsigned integer data types: byte, halfword, word, and doubleword. Unsigned integers consist of only a magnitude field which is the same size as the data type. Unsigned integers are also used to represent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

<u>Data Type</u>	<u>Range</u>
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.



A

86-186

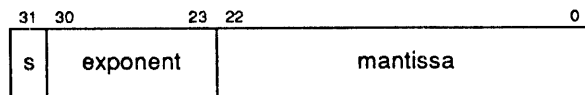
As a fixed length data type, bit data can reside in either a register or in memory. It is also possible to manipulate bit data using the bit string or bit field data types by specifying a length of 1 for these variable length data types.

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

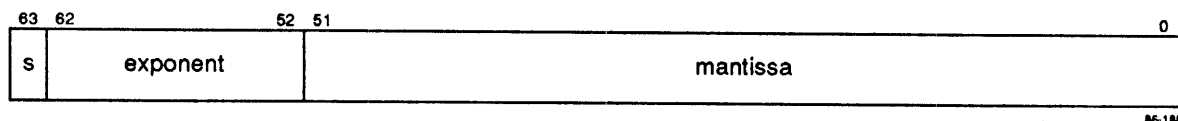
The short real data type is a 32-bit binary floating point representation conforming to the IEEE single precision format. The short real format consists of a mantissa sign bit, an 8-bit biased exponent and a 23-bit mantissa as shown below:



86-187

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:

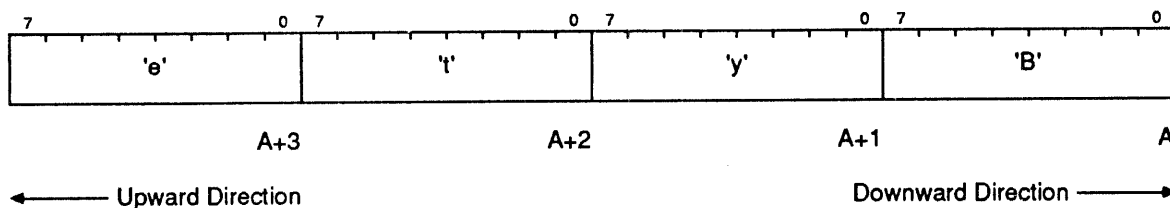
- the sum of the starting address and the length of a character string (in bytes) must be less than $2^{32} - 1$

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.

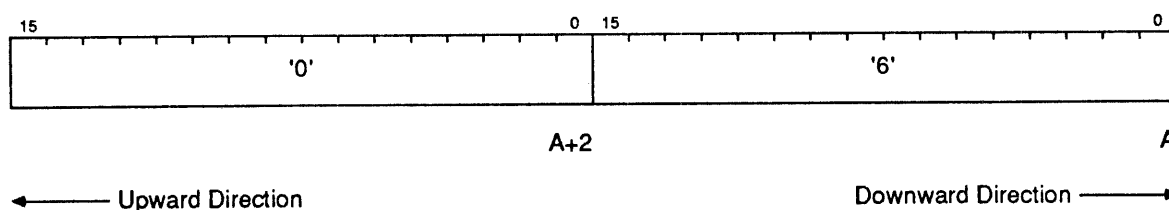
Examples of both byte and halfword character strings are shown below.

Byte character string (N=4)



86-189

Halfword character string (N=2)



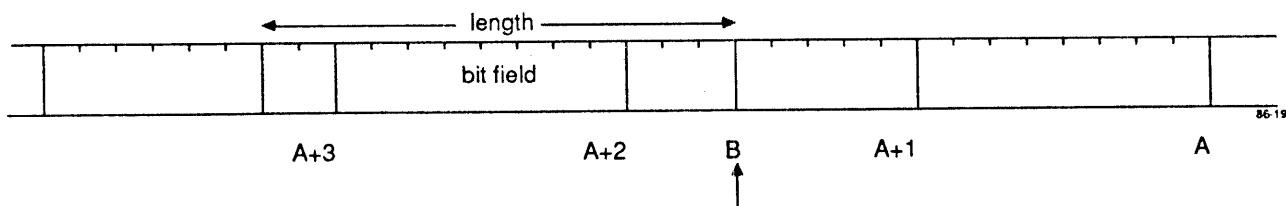
86-190

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



86-191

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

$$\begin{aligned} -2^{(N-1)} &\leq x < 2^{(N-1)} \quad (\text{signed}) \\ 0 &\leq x < 2^{(N-1)} \quad (\text{unsigned}) \end{aligned}$$

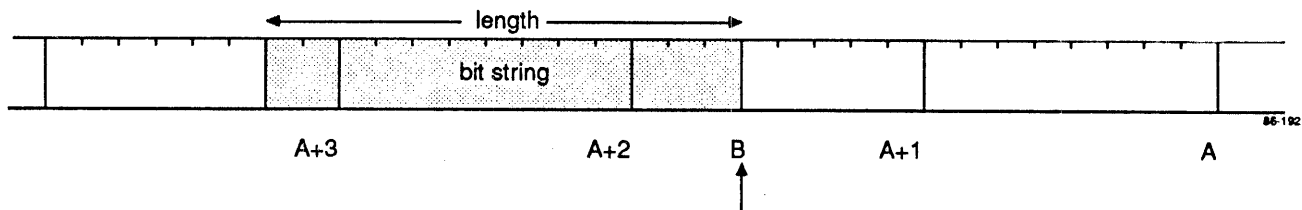
The integer value of a bit field of length 0 is zero.

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

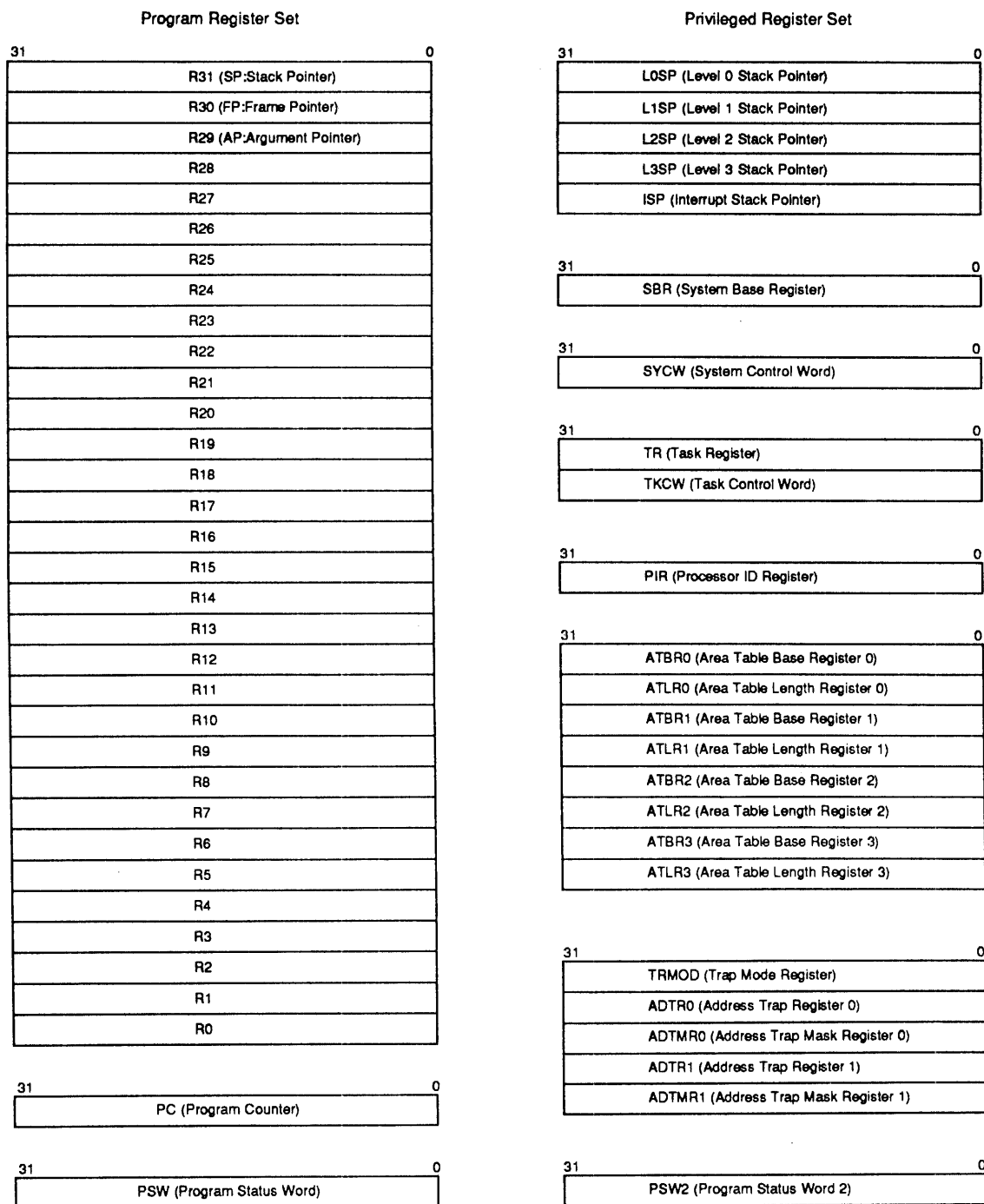
- level 0 – 3 stack pointers.....L0SP – L3SP
- interrupt stack pointer.....ISP

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

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.

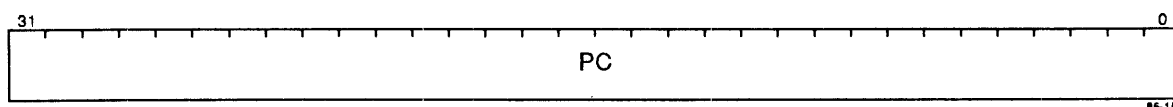
In addition to the AP, FP, and SP registers, other general purpose registers are required by string instructions to allow the instruction to resume following an interrupt or exception. In this case, registers are reserved for use starting from R28 and allocated in a downward direction.

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
```



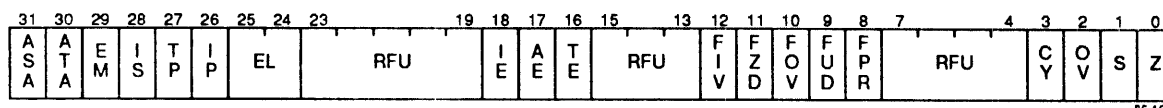
bits 31:0 PC The address of the first byte of the currently executing instruction. The PC contains a virtual address in the virtual address mode, a physical address in the real address mode.

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 occurrence 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