WANG

VS

Principles of Operation

Release 7 Series

VS
Principles of Operation
Release 7 Series

1st Edition — February 1986

715-0422

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

The staff of Wang Laboratories, Inc., has taken due care in preparing this manual. However, nothing contained herein modifies or alters in any way the standard terms and conditions of the Wang purchase, lease, or license agreement by which the product was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the product, the accompanying manual, or any related materials.

SOFTWARE NOTICE

All Wang Program Products (software) are licensed to customers in accordance with the terms and conditions of the Wang Standard Software License. No title or ownership of Wang software is transferred, and any use of the software beyond the terms of the aforesaid license, without the written authorization of Wang, is prohibited.

WARNING

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device, pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

PREFACE

This manual describes the machine architecture of Wang VS systems.

The manual is one of several that, used together, enable the reader to write operating system applications and user applications in assembly language. While the VS Assembly Language Reference (800-1200) describes the conventions of assembly language programming on the VS, the VS Principles of Operation describes the operations of privileged and nonprivileged machine instructions. The VS Operating System Services Reference (715-0423) describes system services and their assembly language calling sequence.

The chapters of this manual fall into three sections:

Chapters 1 through 7 describe machine organization, general instruction and data formats, instruction execution, interrupts, Control mode, and the Debug facility.

Chapter 8 describes the format and operation of each machine instruction.

Chapters 9 through 13 describe the characteristics of I/O devices (including workstations, printers, disk and tape drives), with special attention to input/output command words (IOCWs) and input/output status words (IOSWs).

For information on the VS operating system, the reader can refer to the VS Operating System Services Reference and the current Software Release Bulletin.

The following manuals are referred to in this document:

- VS Data Management System Reference (800-1124)
- VS System Operator's Guide (715-0418)

CONTENTS

CHAPTER 1
INTRODUCTION TO VS SYSTEMS

1.1 VS FAMILY CHARACTERISTICS

The Wang VS computer family consists of medium-scale, general-purpose computers designed to provide sophisticated hardware at a low cost. The general register size on all models is 32 bits. A powerful instruction set has been microprogrammed into the machines, consisting of logical functions, arithmetic instructions (including decimal, floating-point, and decimal floating-point instructions), and queue and push-down stack instructions. This variety of instructions makes for easier programming and faster, more compact code.

Main memory is semiconductor random access memory (RAM) with automatic error correction circuitry. The largest main memory for a VS system is currently 16 MB. For making full use of available memory on any VS system, there is virtual memory support in the form of address translation hardware and several privileged instructions.

Input/output processors or controllers (IOPs or IOCs) optimize central processor (CP) function by governing I/O operations independently of CP activity. The CP and all peripheral processors have direct memory access through main memory controllers. Memory requests are handled according to a priority system and are satisfied on a cycle-stealing basis.

Refer to the following sections for diagrams of particular VS machines, and to Chapter 2 of this manual for a discussion of machine organization.

1.2 VS15 Basic Configuration

The VS15 is the entry-level VS system. The internal CPU data path is 16 bits. The basic configuration of the VS15 consists of the CP (Type 5), main memory, an included 1.2-MB diskette drive, an included fixed-disk drive, and an operator console workstation. Additional I/O devices may be added as options. Other VS systems having the same architecture, including the VS45, may substitute removable-disk drives for the fixed-disk drive.

Figure 1-1 is a diagram of the VS15.

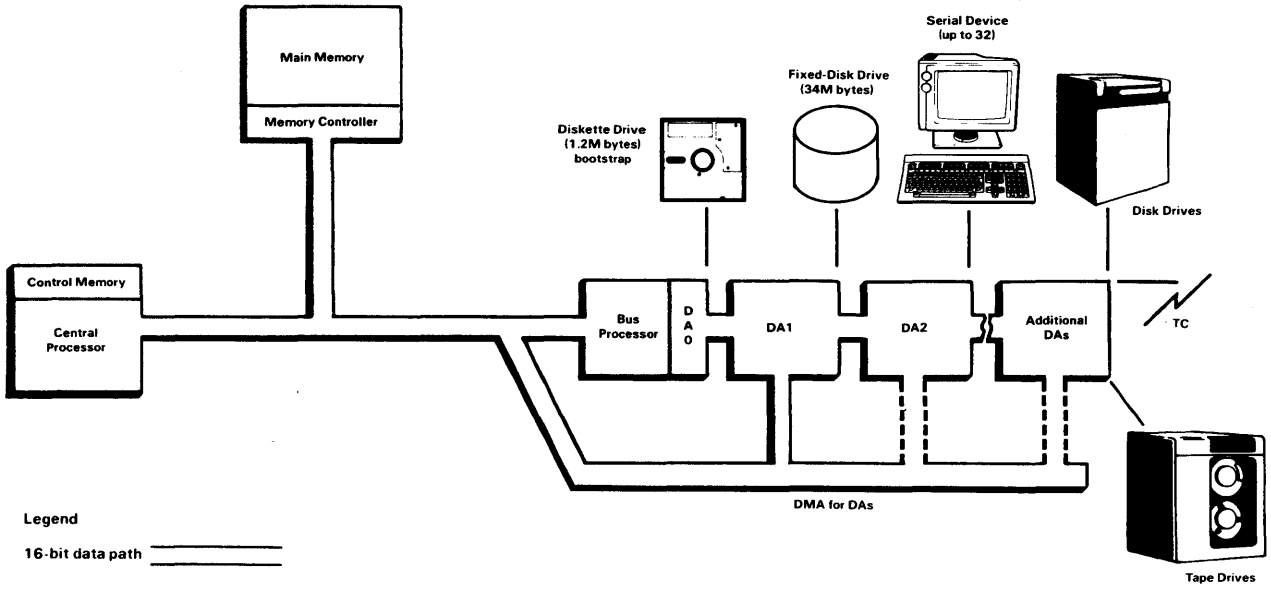


Figure 1-1. VS15 Architecture

1.3 VS65 Basic Configuration

The VS65 is the most powerful of the VS systems utilizing the Bus Processor I/O architecture. The basic configuration of the VS65 consists of the CP (Type 7), main memory and cache memory, a 5 1/4-inch diskette drive, an internal fixed disk of 80 or 160 MB, and an operator console workstation. Additional I/O devices may be added as options.

Figure 1-2 is a diagram of the VS65.

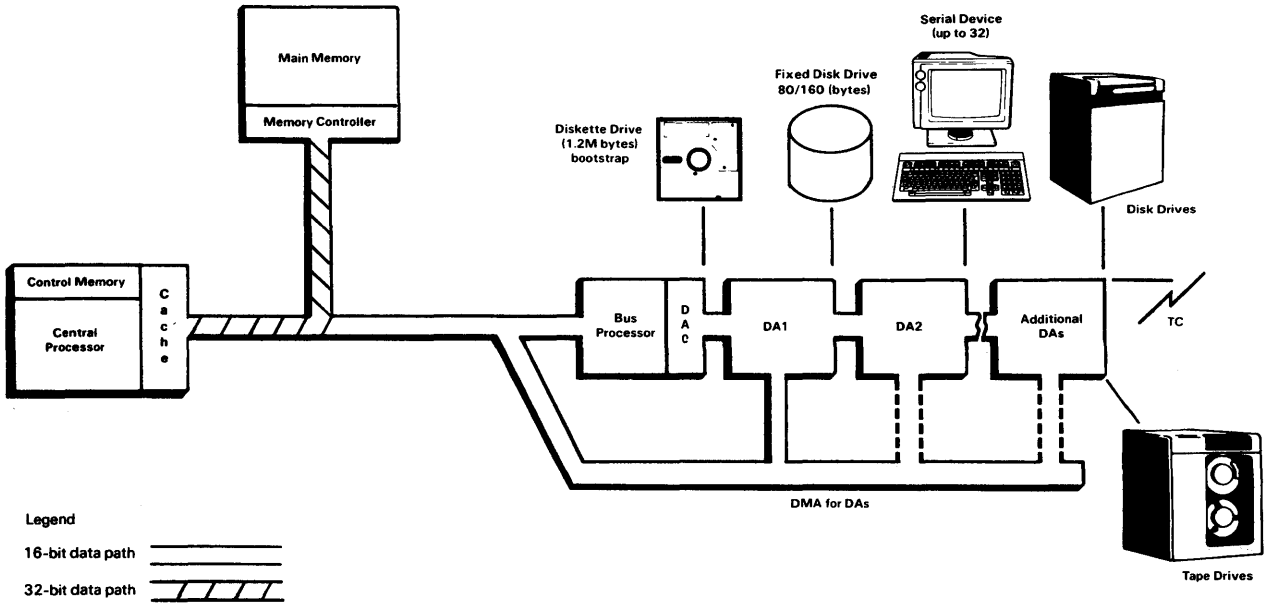


Figure 1-2. VS65 Architecture

1.4 VS100 Basic Configuration

The VS100 supports more main memory, external storage, and peripherals than does the VS65, and executes machine instructions more rapidly. Its internal CPU data path is 32 bits. The basic configuration of the VS100 consists of the CP (Type 4), main memory, one or more removable-disk drives, and an operator console workstation with attached 1.2-MB diskette drive. Additional I/O devices may be added as options. The VS85, VS90, and VS100 share the same central processor type. The VS85 and VS90, however, support only one Bus Adapter. Cache memory is optional on the VS85 and is not available on the VS90.

Figure 1-3 is a diagram of the VS100.

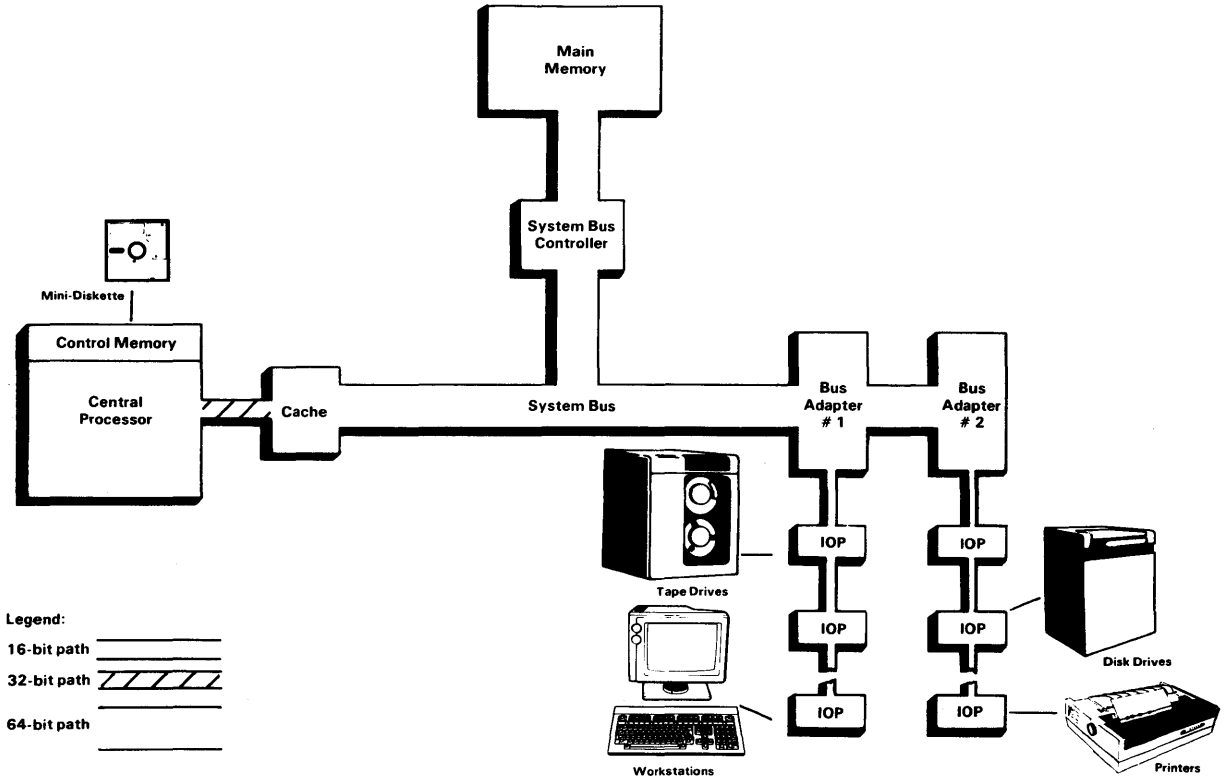


Figure 1-3. VS100 Architecture

1.5 V300 Basic Configuration

The VS300 is the largest and fastest member of the VS family. The basic configuration of the VS300 consists of the CP (Type 8), main memory, one or more removable-disk drives, and an operator console workstation with attached 1.2-MB diskette drive. Additional I/O devices may be added as options. The I/O Controller (IOC) and System Bus Interface (SBI) are similar in function to the I/O Processor (IOP) and Bus Adapter (BA) of the VS100 class systems. The support packet bus provides an SCU-based control path for the system components.

Figure 1-4 is a diagram of the VS300.

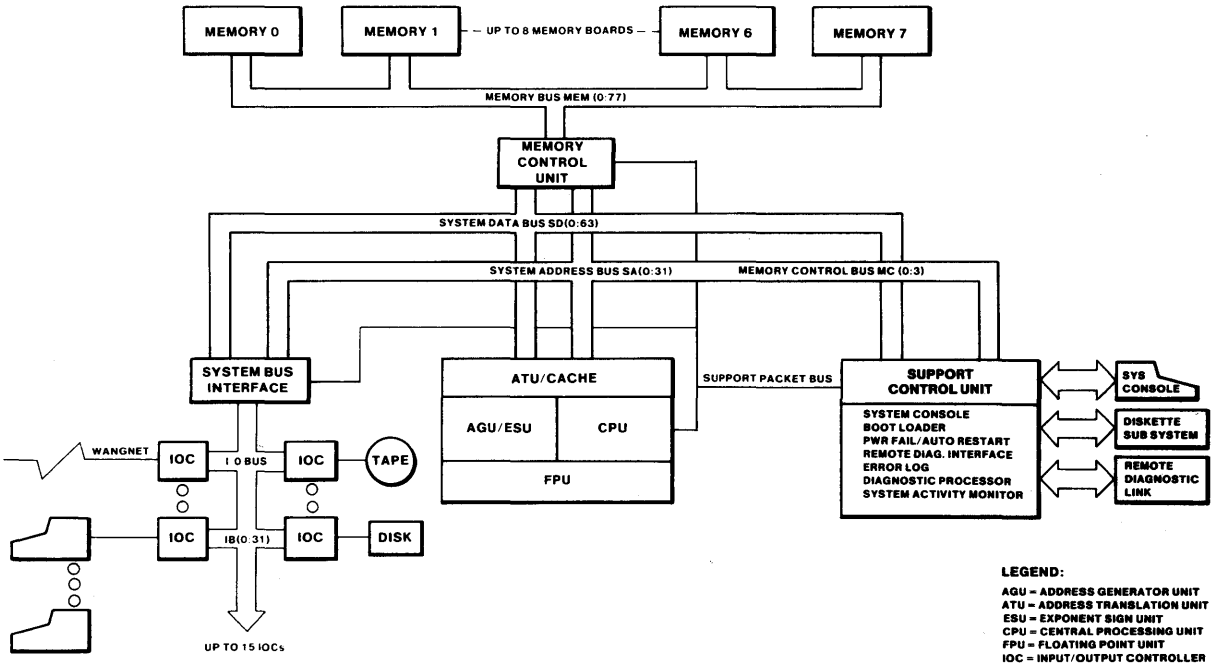


Figure 1-4. VS300 Architecture

FIGURES

FIGURES (continued)

TABLES

CHAPTER 2
MACHINE ORGANIZATION

2.1 CENTRAL PROCESSOR

The Central Processor (CP) contains facilities for addressing main memory, for fetching and storing information, for arithmetic and logical processing of data, for sequencing instructions in the desired order, and for initiating communication between memory and external devices.

2.1.1 General Registers

The processor can address information in 16 general registers. The general registers may be used as index registers in address arithmetic and indexing, and as accumulators in fixed-point arithmetic and logical operations. The registers have a capacity of one word (32 bits). The general registers are identified by Numbers 0-15 and are specified by a 4-bit R field in an instruction format. Some instructions have several R fields.

2.1.2 Floating-Point Registers

There are four floating-point registers, specified as Registers 0, 2, 4, and 6. Each such register is 64 bits in length and can contain one floating-point number. These registers are addressed by the floating-point and decimal floating-point instructions only.

2.1.3 Control Registers

The control registers provide a means of maintaining and manipulating control information that resides outside the Program Control Word (PCW).

Sixteen 32-bit registers are provided for control purposes. These registers are not part of addressable storage. The instruction LOAD CONTROL (LCTL) provides a means of loading control information from main memory into control registers, while STORE CONTROL (STCTL) permits information to be transferred from control registers to main memory. These instructions operate in a manner similar to LOAD MULTIPLE and STORE MULTIPLE. Also, the JUMP TO SUBROUTINE ON CONDITION INDIRECT (JSCI), RETURN ON CONDITION (RTC), SUPERVISOR CALL (SVC), and SUPERVISOR CALL EXIT (SVCX) instructions modify Control Register 1. LCTL and SVCX are privileged instructions.

At the time the registers are loaded, the information is not checked for exceptions, such as addresses designating unavailable locations. The validity of the information is checked, and the exceptions, if any, are indicated, at the time that the information is used. Control register allocations for the VS systems are as follows:

Control Register	Allocation
CR0	Reserved
CR1	Save area back chain
CR2	System stack limit word
CR3	Debug table descriptor
CR4	Prior instruction address
CR5	Current instruction address; debug table inspection flag
CR6-7	Linkage table descriptors
CR8	Pointer to active stack header block
CR9	Pointer to stack header table
CR10	Debug scope
CR11	Reserved
CR12-13	Time of day clock
CR14-15	Clock comparator

This section provides only a general description of register contents and functions. For more detailed information, refer to the facility or instruction with which the registers are associated.

Control Register 1 is updated by the JSCI, RTC, SVC, and SVCX instructions to maintain a protected back chain of program calls and supervisor service entries (supervisor calls). Control Register 2, referred to as the system stack limit word, is used to detect stack overflow. Control Register 3 contains the Debug table address and a count of entries in the table. Control Registers 4-5 are used by the Debug Facility for storing trapped instructions; also, a flag-byte in Control Register 5 synchronizes processing of the Debug table. Control Registers 6-7 describe the location, length, and element-size of a task's linkage table. This table is referred to during execution of the JSCI instruction, as explained in Section 4.6.2. The structures pointed to by Control Registers 8-9 are referred to by the JSCI instruction when branching to a dynamically-linked subroutine. Control Register 10 holds two Debug scope fields that demarcate the range of process levels wherein Debug traps are taken. Registers 12-15 are associated with the clock.

2.1.4 Translation RAM (T-RAM)

Translation RAM (T-RAM) is a section of local CP memory (RAM) on the VS15, VS65, and VS100 systems, consisting of halfword entries. A field in each entry indicates the physical page frame wherein a virtual page has been loaded.

The number of T-RAM entries and the size of the physical page frame field varies between classes of VS systems. The number of entries in T-RAM determines the maximum amount of virtual address space that can be mapped to physical main memory; and the size of the physical page frame field determines the maximum amount of physical main memory supported on the system.

The size of T-RAM on VS15, VS65, and VS100 systems is shown below, along with the maximum size of virtual and physical memory supported.

VS15

Number of T-RAM entries:	4K
Total size of T-RAM:	8 KB
Size of Page Frame field:	10 bits
Maximum Virtual Memory:	8 MB
Maximum Physical Memory:	2 MB

VS65

Number of T-RAM entries:	8K
Total size of T-RAM:	16 KB
Size of Page Frame field:	11 bits
Maximum Virtual Memory:	16 MB
Maximum Physical Memory:	4 MB

VS 100

Number of T-RAM entries:	4K (8K optional)
Total size of T-RAM:	8 KB (16KB optional)
Size of Page Frame field:	13 bits
Maximum Virtual Memory:	8 MB (16MB optional)
Maximum Physical Memory:	16 MB

2.1.5 T-RAM Monitor Area

The T-RAM monitor area is used for selective clearing of the T-RAM entries. Each monitor area entry records the number of a virtual page mapped to physical memory. This number serves as an index to a T-RAM entry that must be cleared at the end of a task's time slice. The monitor area itself is cleared by the monitor area function of RESET REFERENCE AND CHANGE BITS (RRCB) instruction. The monitor area of the VS15 and VS100 is in local CP memory; that of the VS65 is in main memory. Further information on monitor areas is provided in Section 4.3.7.

2.1.6 Translation Buffer (T-BUF)

The Translation Buffer (T-BUF) is a section of local CP memory on the VS300 consisting of 1K entries. A field in each entry indicates the physical page frame wherein a virtual page has been loaded.

Whereas there is one T-RAM entry for one page of virtual address space, there is one T-BUF entry for eight virtual pages. Since more than one virtual page number can map to a single entry, each T-BUF entry has a tag associated with it. Tags are stored in a separate area of local memory, called a tag store area and consisting of 1K 3-bit entries.

2.1.7 Reference and Change Table (RCT)

The RCT is an area of local memory that contains two bits per page frame (2048 bytes on a 2048-byte boundary) of physical memory. Whenever a location in a page frame is referenced by a machine instruction, the reference bit of the corresponding RCT entry is set to 1. When this reference involves modification of the memory location, the change bit in the RCT is also set to 1. RCT entries are cleared by the RESET REFERENCE AND CHANGE BITS (RRCB) instruction.

2.1.8 Arithmetic and Logic Unit (ALU)

The arithmetic and logic unit can process binary integers of fixed length, decimal integers of variable length, and logical information of either fixed or variable length.

Arithmetic and logical operations performed by the CP fall into five classes: fixed-point arithmetic, floating-point arithmetic, decimal arithmetic, decimal floating-point arithmetic, and logical operations. These classes differ in the data formats used, the registers involved, the operations provided, and the way the field length is stated.

2.2 CLOCK

2.2.1 Time-of-Day Clock

The time-of-day clock provides a consistent measure of elapsed time suitable for the indication of date and time. The clock is a double-register, 64-bit binary counter. Time is measured by incrementing the value of the clock, following the rules for unsigned fixed-point arithmetic. The clock is incremented by adding 1 to the low-order bit position at split line frequency (1/100 or 1/120 second) for the VS15 and VS65 class systems, 2.5 MHz for the VS100 and VS300 class systems. The resolutions of these clocks are, respectively, 17 milliseconds and 400 nanoseconds.

When the incrementing of the clock causes a carry to be propagated out of bit position 0, the carry is ignored and counting continues from zero. The program is not alerted, and no interruption condition is generated as a result of the overflow. The clock runs while the machine is powered on, even when the machine is in Control mode or wait state.

The clock value resides in Control Registers 12 and 13 and is set to zero during a power-up. This value can be manipulated under program control by means of the LOAD CONTROL and STORE CONTROL instructions.

2.2.2 Clock Comparator

The clock comparator provides a means of causing an interruption when the time-of-day clock has passed a value specified by the program. The clock comparator has the same format as the time-of-day clock.

The clock comparator value is compared with the value of the time-of-day clock, each being regarded as a 64-bit unsigned number. Whenever the time-of-day clock value is greater than or equal to the value of the clock comparator, and clock interrupts are enabled (PCW T-bit=1), a clock interrupt is generated. The value of the clock comparator resides in Control Registers 14 and 15 and is set to all 1s during power-up. An interruption request disappears if the value in Control Registers 12 and 13 or Control Registers 14 and 15 is changed so that the value in Control Registers 12 and 13 is less than that in Control Registers 14 and 15. These values can be manipulated under program control by means of the LOAD CONTROL and STORE CONTROL instructions.

2.3 I/O PROCESSORS (IOPs)

Input/output processors (IOPs) connect the CP and main memory with the input/output (I/O) devices. IOPs relieve the CP of the burden of communicating directly with I/O devices and permit data processing to proceed concurrently with I/O operations. IOPs provide the logical capabilities necessary to operate and control I/O devices. IOPs decode the commands fetched from main memory and interpret them for particular devices.

For the VS15 and VS65, a single bus processor (BP) controlling several device adapters (DAs) does the work of VS100 IOPs. For the VS100, intelligent bus adapters (BAs) provide an interface between the CP and IOPs. The input/output controllers (IOCs) and system bus interfaces (SBIs) of the VS300 are functionally equivalent, respectively, to IOPs and BAs.

2.4 MAIN MEMORY

Main memory for all VS systems consists of semiconductor random access memory (RAM) with automatic error correction circuitry. Memory is automatically refreshed by hardware at intervals of 10 msec and therefore cannot be maintained past system power-off. Requests for memory access by the CP and other processors are handled on a priority system (whereby the CP has lowest priority) and are satisfied on a cycle-stealing basis. Therefore, instructions that fetch and subsequently store data do not necessarily use consecutive memory cycles, because one or more intervening cycles may be devoted to I/O operations.

2.4.1 Information Formats

VS systems transmit information between main memory and the CP in logical units of eight bits or a multiple thereof. Each 8-bit unit of information is called a byte, the basic building block of all formats. All storage capacities are expressed in terms of the number of bytes provided.

Bytes may be handled separately or grouped together in fields. The address of any field or group of bytes is the address of its leftmost byte. A word is a field of four consecutive bytes whose address is a multiple of 4. A doubleword is a field of two consecutive words whose address is a multiple of 8, and a halfword is a field of two consecutive bytes whose address is a multiple of 2.

For the VS15, byte-aligned write operations of one or two bytes and halfword-aligned read operations of two bytes are supported. For the VS65, halfword-aligned read and write operations of one or two bytes are supported. For the VS100 and VS300, byte-aligned write operations of one, two, four, or eight bytes are supported, along with doubleword-aligned read operations of eight bytes only.

In any instruction format or any fixed-length operand format, the bits or bytes making up the format are consecutively numbered from left to right starting with 0, and are indicated in the line under the format description. Figure 2-1 is a diagram of these units of information.

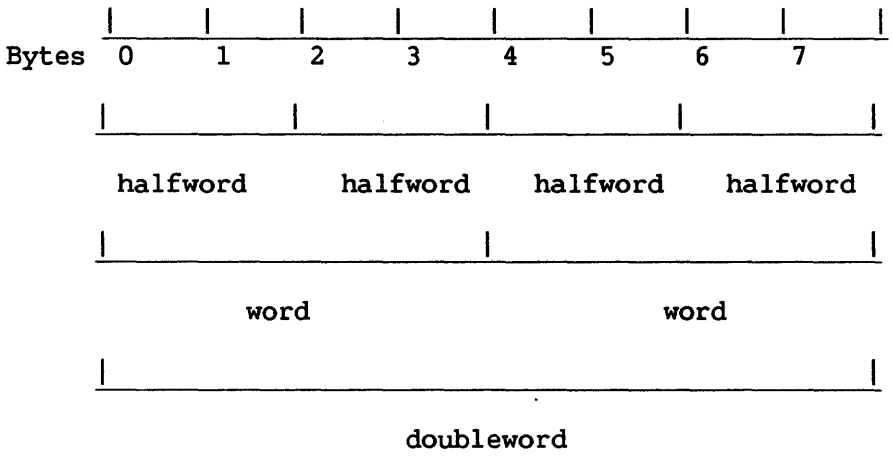


Figure 2-1. Sample Information Formats

2.4.2 Conventions of Description

To indicate the left or right end of any field or word definition, the following terminology and abbreviations are used throughout this manual:

Leftmost Portion	Rightmost Portion
Most Significant Byte (MSB)	Least Significant Byte (LSB)
High Order	Low Order
Most Significant bit (MSb)	Least Significant bit (LSb)

2.4.3 Addressing

Byte locations in memory are numbered consecutively, starting with 0; each number is considered the address of the corresponding byte. A group of bytes in memory is addressed by the leftmost byte of the group. The number of bytes in the group is either implied or explicitly defined by the operation. The VS addressing arrangement uses a 24-bit virtual address to accommodate a maximum of 16,777,216 byte addresses.

When only a part of the maximum storage capacity is available in a given installation, the available storage is normally a contiguous range of physical addresses starting at Address 0. An addressing exception is recognized when any part of an operand is located beyond the maximum available capacity of an installation. The addressing exception is recognized when the data is used and causes a program interruption.

Refer to Sections 4.2 and 4.3 for details of addressing and address translation.

CHAPTER 3
DATA ORGANIZATION

3.1 INSTRUCTIONS

VS machine instructions perform five classes of operations: fixed-point arithmetic, floating-point arithmetic, decimal arithmetic, decimal floating-point arithmetic, and logical operations.

Each instruction consists of two major parts: an operation code, which specifies the operation to be performed, and the designation of the operands that participate.

3.1.1 Operands

Operands can be grouped in three classes: operands located in registers, immediate operands, and operands in main memory. Operands may be either explicitly or implicitly designated.

Register operands can be located in general, floating-point, or control registers, and are specified by identifying the register in a 4-bit field, called the R field, in the instruction. For some instructions, an operand is located in an implicitly designated register.

Immediate operands are contained within the instruction, and the 8-bit field containing the immediate operand is called the I field.

The length of operands in main memory may be either implied or specified by a 4-bit mask in the instruction called the M field. The length of operands in main memory may also be specified by a 4-bit or 8-bit length parameter, called the L field, in the instruction.

The addresses of operands in main memory are specified by a format that uses the contents of a general or base register as part of the address. The address in the general register is called the B field and the additional displacement address (which may be 0) is the D field. The X field denotes an address in an index register, which is added to the base register address. A detailed explanation of the B, D, and X fields is given in Section 4.2.1.

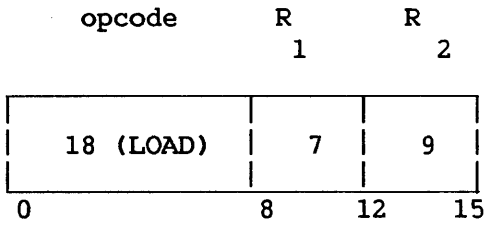
Operands on a stack in main memory may be specified by a 4-bit field in the instruction, called the S field, that specifies the user stack vector or implies the system stack vector. A stack vector comprises a pair of registers pointing to the stack limit and the stack top, as explained in Section 3.9.

The VS allows up to three operands, depending on the instruction format. For purposes of describing the execution of instructions, operands are designated as first, second, and third operands. The operand to which a field in an instruction format applies is generally denoted by the number following the code name of the field (e.g., R1, B1, L2, D2).

In general, two operands participate in an instruction execution, and the result replaces the first operand. An exception is instructions with STORE in the name, where the result replaces the second operand. Except for storing the final result, the contents of all registers and memory locations that participate in the addressing or execution part of an operation remain unchanged.

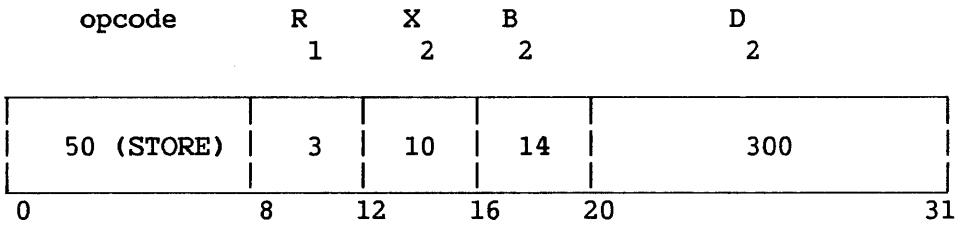
As an aid in describing the logic of the instruction format, examples showing the format and and operation of two instructions are given:

RR Format: LR 7,9



Execution of the LOAD instruction copies the contents of General Register 9 to General Register 7.

RX Format: ST 3,TOTAL



Execution of the STORE instruction stores the contents of General Register 3 at a main memory location addressed by the sum of 300 and the contents of General Registers 14 and 10, with the data name TOTAL.

3.1.2 Instruction Format

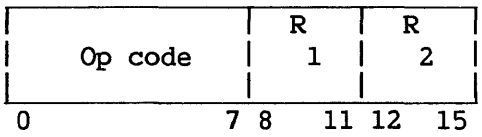
An instruction is one, two, three, or four halfwords in length and must be located in main memory on an integral halfword boundary.

The nine basic instruction formats are denoted by the format codes RL, RR, RRL, RX, RS, SI, S, SS, and SSI. The format codes express, in general terms, the operation to be performed.

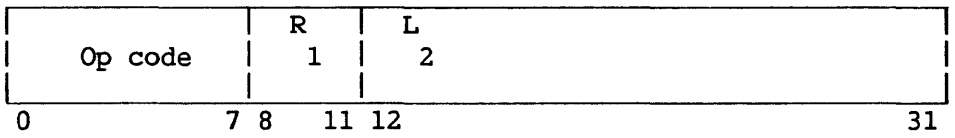
- RR - Register-to-register operation
- RL - Register-to-register (relative) operation
- RX - Register-and-indexed-storage operation
- RS - Register-and-storage operation
- RRL - Register-to-storage (relative) operation
- SI - Storage-and-immediate-operand operation
- S - Implied-operand-and-storage operation
- SS - Storage-to-storage operation
- SSI - Storage-to-storage-and-immediate-operand operation

The following diagrams illustrate the representation of the nine instruction formats in VS memory.

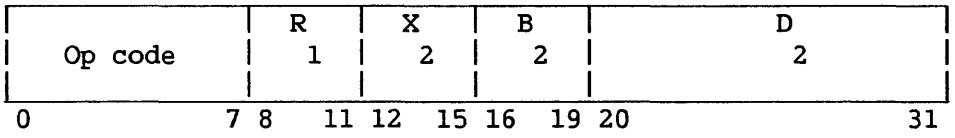
RR Format--One Halfword



RL Format--Two Halfwords

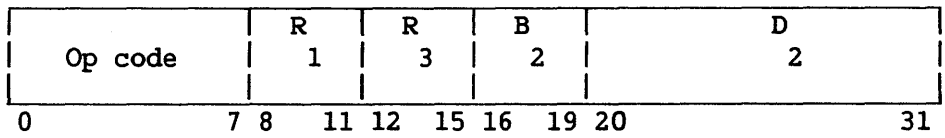


RX Format--Two Halfwords

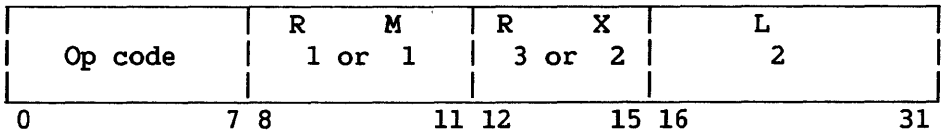


Programming Note:
If D_2 and B_2 are omitted in RX format, R_1 is used for D_2.

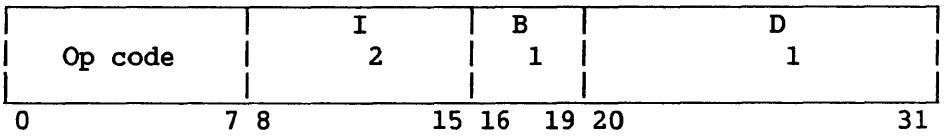
RS Format--Two Halfwords



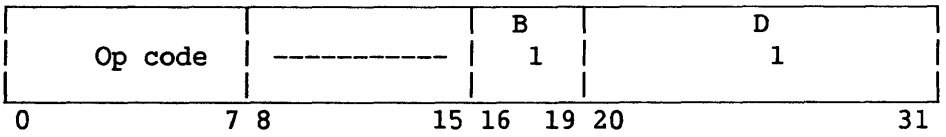
RRL Format--Two Halfwords



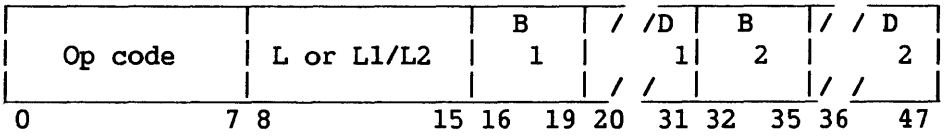
SI Format--Two Halfwords



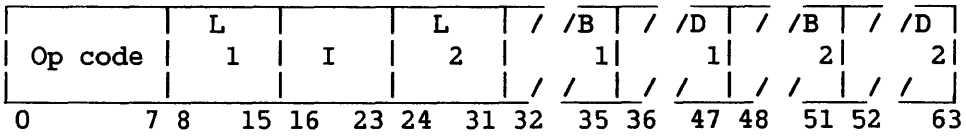
S Format--Two Halfwords



SS Format--Three Halfwords



SSI Format--Four Halfwords



3.1.3 Operation Code

The first byte of an instruction contains the operation code (op code). The length and format of an instruction are specified by the first two bits of the operation code.

Bit Positions 0 and 1	Instruction Length	Instruction Format
00	Halfword	RR
01	Two halfwords	RX
10	Two halfwords	RS, SI, S, RL, or RRL
11	Three or four halfwords	SS or SSI

3.2 FIXED-POINT INSTRUCTIONS

The binary fixed-point instructions perform binary arithmetic on operands serving as addresses, index quantities, and counts, as well as on fixed-point data. In general, both operands are to be considered unsigned and 24 bits long for address computations, or signed and 31 or 15 bits long for arithmetic computations. One operand is always in one of the 16 general registers; the other operand may be in main memory or in a general register.

The binary fixed-point instructions provide for loading, adding, subtracting, comparing, multiplying, dividing, and storing, as well as for the radix conversion and shifting of fixed-point operands.

The condition code is set as a result of all ADD, SUBTRACT, COMPARE, and SHIFT operations.

3.2.1 Data Format

Binary fixed-point data in main memory occupies a 16-bit halfword or a 32-bit word. This data must be located on integral boundaries for these units of information; that is, halfword or fullword operands must be addressed with one or two low-order address bits set to 0, respectively.

Fixed-point numbers occupy a fixed-length format consisting of an integer field. This format is shown in Figure 3-1. When held in one of the general registers, a fixed-point quantity occupies all 32 bits of the register. In register-to-register operations the same register may be specified for both operand locations.

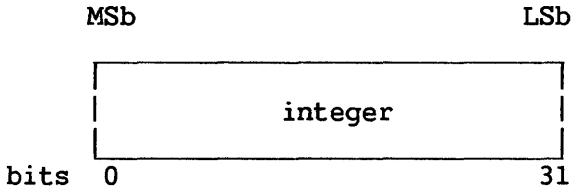


Figure 3-1. Fixed-Point Fullword Data Format

3.2.2 Fixed-Point Arithmetic

The basic arithmetic operands are the 32-bit fixed-point binary word and the 16-bit fixed-point binary halfword.

Fixed-point arithmetic can be used both for integer operand arithmetic and for address arithmetic. This combined usage permits the entire fixed-point instruction set and several logical operations to be used in address computation. Thus, multiplication, shifting, calculation, and logical manipulation of address components are possible.

Additions, subtractions, multiplications, and comparisons are performed upon one operand in a register and another operand either in a register or in memory. A word in a register may be shifted left or right. A pair of conversion instructions -- CONVERT TO BINARY (CVB) and CONVERT TO DECIMAL (CVD) -- provide for translation between decimal and binary number bases without the use of tables. Multiple-register LOAD and STORE instructions facilitate subroutine switching.

In an unsigned fixed-point number, all bits may be considered to express the absolute value of the number. Only the AL, SL, and CL instructions take signed binary operands; all three require fullword operands.

A fixed-point number may also be considered a signed quantity, where the leftmost bit represents the sign, followed by the 31-bit or 15-bit integer field. Positive numbers are then represented in true binary notation with the sign bit set to 0, and negative numbers in 2's-complement notation with a 1 in the sign-bit position.

The 2's-complement representation of a negative number may be considered the sum of the integer part of the field, taken as a positive number, and the maximum negative number. The 2's complement of a number is obtained by inverting each bit of the number and adding a 1 in the low-order bit position. A negative zero is not included in 2's-complement notation; so, the set of negative numbers is one larger than the set of positive numbers.

3.3 DECIMAL INSTRUCTIONS

Decimal instructions allow arithmetic, shifting, and editing operations on decimal data.

3.3.1 Decimal Arithmetic

Decimal arithmetic lends itself to procedures that require few computational steps between the source input and the output. This type of processing is frequently found in commercial applications, particularly those using problem-oriented languages. Because of the limited number of arithmetic operations performed on each item of data, radix conversion from decimal to binary and back to decimal is not justified, and the use of registers for intermediate results yields no advantage over storage-to-storage processing. Hence, decimal arithmetic is provided, and both operands and results are located in memory. Decimal arithmetic includes addition, subtraction, multiplication, division, and comparison.

Decimal arithmetic operates on data in the packed format. In this format, two decimal digits are placed in each 8-bit byte. Each digit is interpreted as an integer and is right-aligned in its 4-bit field. A decimal number is kept in true notation with a sign in the least significant 4-bit field of the string of bytes that compose the number.

Processing takes place from right to left between main memory locations. All decimal arithmetic instructions use a two-address format. Each address specifies the leftmost byte of an operand. Associated with this address is a length field, which indicates the number of additional bytes that the operand extends beyond the first byte.

The decimal arithmetic instructions provide for addition, subtraction, comparison, multiplication, and division, as well as format conversion of variable-length operands.

The condition code is set as a result of all decimal instructions except MP and DP.

The sign of the result is determined by the rules of algebra. When an operation (other than PACK AND ALIGN (PAL)) is completed without an overflow, a zero sum result has a positive sign, but when high-order digits are lost because of an overflow, a zero result may be either positive or negative, as determined by what the sign of the correct result would have been. A decimal instruction will set the condition code even if a decimal overflow exception occurs.

3.3.2 Data Formats

Decimal operands reside in main memory only. They occupy fields that may start at any byte address and are composed of from one to sixteen 8-bit bytes.

Lengths of the two operands specified in an instruction need not be the same. If necessary they are considered to be extended with 0s to the left of the most significant digits. Results never exceed the limits set by address and length specification. Lost carries or lost digits from arithmetic operations are signaled as decimal overflow exceptions.

Packed Decimal Number

In the packed format, numbers are represented as right-aligned true integers, with a plus or minus sign in the rightmost four bit positions.

The decimal digits 0-9 are represented in the 4-bit binary-coded-decimal form by 0000-1001. The codes 1010-1111 represent signs rather than digits, as shown in Table 3-1. The preferred sign codes are generated by all decimal arithmetic instructions.

Table 3-1. Bit Codes for Digits and Signs

Digit	Code	Preferred Sign Code	Allowed Sign Code
0	0000	- 1101	- 1101
1	0001	+ 1111	+ all other codes
2	0010		
3	0011		
4	0100		
5	0101		
6	0110		
7	0111		
8	1000		
9	1001		

All decimal arithmetic is performed on data in the packed format. In the packed format, two decimal digits are adjacent in a byte, except for the rightmost byte of the field. In the rightmost byte a sign is placed to the right of the decimal digit. Both digits and a sign are encoded and occupy four bits each.

Decimal operands and results are represented by 4-bit binary-coded-decimal digits packed two to a byte. They appear in fields of variable length and are accompanied by a sign in the rightmost four bits of the least significant byte, as shown in Figure 3-2. Operand fields may be located on any byte boundary, and may have a length of up to 31 digits and a sign. Operands participating in an operation may have different lengths. Packing of digits within a byte and of variable-length fields within memory results in efficient use of memory, increased arithmetic performance, and an improved rate of data transmission between memory and files.

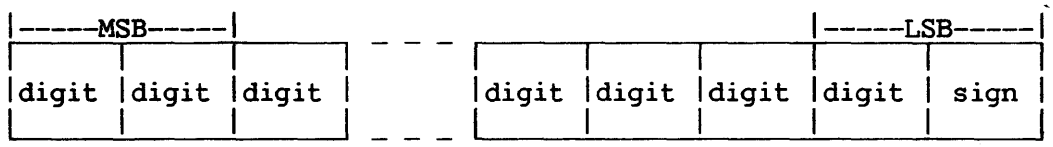


Figure 3-2. Packed Decimal Number Format

Zoned Decimal Number

A zoned decimal number is a right-aligned integer with one digit code per byte and the sign code in the four high-order bits of the low-order byte, as shown in Figure 3-3. Zoned decimal numbers are converted to packed format by the PACK instruction. The four high-order bits (zone bits) of bytes other than the low-order byte do not affect the resulting packed decimal number.

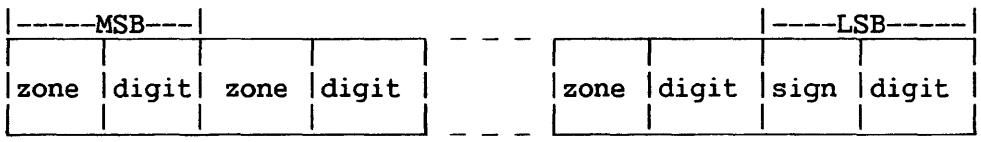


Figure 3-3. Zoned Decimal Number Format

External Decimal Number

Decimal numbers may also appear in an external format as a subset of the 8-bit alphameric character set. External decimal format is shown in Figure 3-4. This representation is required for character-set-sensitive I/O devices. An external format number carries its sign as an 8-bit ASCII character that precedes or follows the ASCII number. The external format is not used in decimal arithmetic operations. The PAL and PACK instructions are provided to transform external data into packed data, and the ED, EDMK, UNPACK, UNPAL, and UNPU instructions may be used to change data from packed to external format.

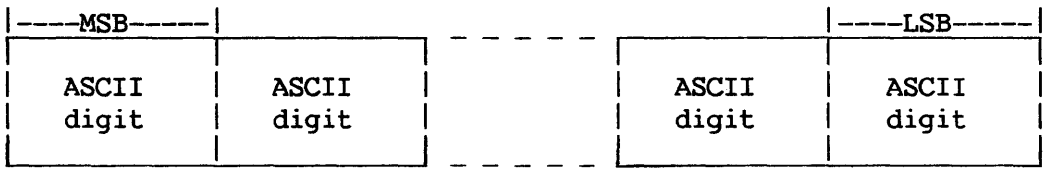


Figure 3-4. External Decimal Number Format

The sign character may appear as the first or the last character in the external format character string. The external format string for any field that is to be converted to a packed format field cannot exceed 16 ASCII characters.

The fields specified in decimal instructions either should not overlap at all or should have coincident rightmost bytes. In ZERO AND ADD, the destination field may also overlap to the right of the source field. Because the code configurations for digits and sign are verified during arithmetic, improperly overlapping fields are recognized as data exceptions.

The rules for overlapped fields are established for the case where operands are fetched right to left from memory, eight bits at a time, just before they are processed. Similarly, the results are placed in memory eight bits at a time, as soon as they are generated.

3.4 FLOATING-POINT INSTRUCTIONS

The floating-point instruction set is used to perform calculations on operands with a wide range of magnitudes. Floating-point operations yield results scaled to preserve precision.

3.4.1 Floating-Point Arithmetic

A floating-point number consists of a signed exponent and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess-64 binary notation; the fraction is expressed as a hexadecimal number having a radix point to the left of the high-order digit.

To avoid unnecessary storing and loading operations for results and operands, four floating-point registers are provided. The floating-point instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, storing, and sign control, of both long and short operands. Operations may be either register-to-register or storage-to-register.

Maximum precision is preserved in addition, subtraction, multiplication, and division by producing normalized results. For addition, instructions are also provided that generate unnormalized results. Normalized and unnormalized operands may be used in any floating-point operation. Normalization is discussed in Section 3.4.3.

The condition code is set as a result of all floating-point sign control, add, subtract, and compare operations. Multiplication, division, loading, and storing leave the code unchanged. The condition code can be used for decision-making by subsequent branch-on-condition instructions. The condition code can be set to reflect two types of results for floating-point arithmetic. For most operations, the Codes 0, 1, and 2 indicate, respectively, that the result is 0, less than 0, or greater than 0. A zero result is indicated whenever the result fraction is 0, including a forced 0. Code 3 is never set by floating-point operations.

In comparisons, the States 0, 1, and 2 indicate, respectively, that the first operand is equal, low, or high.

3.4.2 Data Format

Floating-point data appears in a fixed-length format that may be either 8-byte (long) or 4-byte (short), as pictured in Figure 3-5. Operands in either format may be specified either in main storage or in floating-point registers. The floating-point registers are numbered 0, 2, 4, and 6.

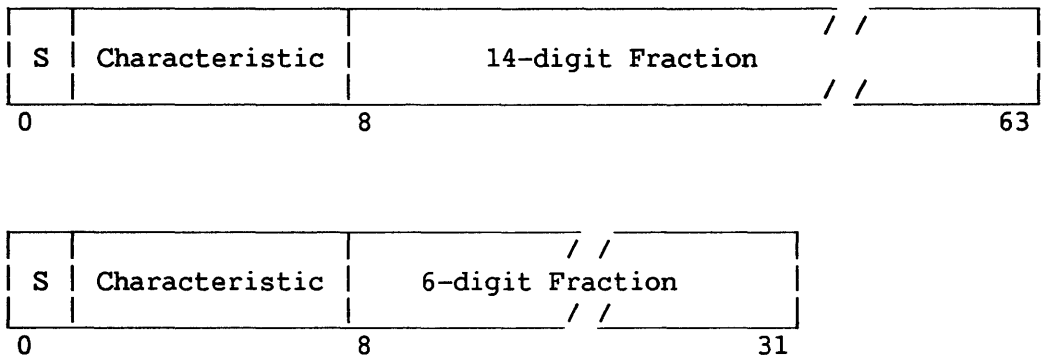


Figure 3-5. Long and Short Floating-Point Numbers

The first bit is the sign bit (S). The subsequent seven bit positions are occupied by the characteristic. The fraction field has either 14 or 6 hexadecimal digits, for long or short floating-point numbers, respectively.

Short floating-point numbers occupy only the leftmost 32 bit positions of a floating-point register. When a floating-point register is used as the source of a short floating-point number, the rightmost 32 bit positions of the register are ignored. When a floating-point register is used as the destination of a short floating-point number, the rightmost 32 bit positions of the register remain unchanged.

The entire set of floating-point functions is available for both short and long operands. These instructions generate a result that has the same format as the sources, except that in the case of the ME and MER multiply instructions, a long product is produced from a short multiplier and short multiplicand. The LOAD ROUNDED instruction provides for rounding from long to short format, while the LOAD SHORT TO LONG instruction provides for expansion from short to long format.

Although final results have either 14 or 6 fraction digits, intermediate results in ADD NORMALIZED, SUBTRACT, ADD UNNORMALIZED, COMPARE, HALVE, and MULTIPLY may have one additional low-order digit. This low-order digit, the guard digit, increases the precision of the final result.

The fraction of a floating-point number is expressed in hexadecimal digits. The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. To provide the proper magnitude for the floating-point number, the fraction is considered to be multiplied by a power of 16. The characteristic, Bits 1-7 of both long and short floating-point formats, indicates this power. The bits within the characteristic field can represent numbers from 0 through 127. To accommodate large and small magnitudes, the characteristic is formed by adding 64 to the actual exponent. The range of the exponent is thus -64 through +63. This technique produces a characteristic in excess-64 notation.

Both positive and negative quantities have a true fraction, the difference in sign being indicated by the sign bit. The number is positive or negative, accordingly, as the sign bit is 0 or 1.

The allowed range of magnitude (M) is 16**-65 < M < (1-16**-14) * 16**63 for a long floating-point number, and 16**-65 < M < (1-16**-6) * 16**63 for a short floating-point number; or approximately 5.4 * 10**-79 < M < 7.2 * 10**75 in both formats.

A number with a characteristic of 0, a fraction of 0, and a plus sign is called a true 0. A true 0 may result from an arithmetic operation because of the particular magnitude of the operands. A result is forced to be true 0 when one of the following conditions occur:

1. An exponent underflow occurs and the exponent-underflow mask (PSW Bit 38) is 0.

2. A result fraction of an addition or subtraction operation is 0 and the significance mask (PSW Bit 39) is 0.

3. The operand of HALVE, one or both operands of MULTIPLY, or the dividend in DIVIDE has a fraction of 0. When a program interruption due to exponent underflow occurs, a true 0 fraction is not forced; instead, the fraction and sign remain correct and the characteristic is too large by 128.

When a program interruption due to lost significance occurs, the fraction remains 0 and the sign and characteristic remain correct. Whenever a result has a fraction of 0, the exponent overflow and underflow exceptions do not cause a program interruption. When a divisor has a fraction of 0, division is suppressed, a floating-point divide exception exists, and a program interruption occurs. In addition and subtraction, an operand with a fraction or characteristic of 0 participates as a normal number.

The sign of a sum, difference, product, or quotient with a fraction of 0 is positive.

3.4.3 Normalization

A quantity can be represented with the greatest precision by a floating-point number when that number is normalized, that is, when the nonzero fraction digits are shifted left as far as possible so that the exponent is of the minimum possible magnitude. A normalized floating-point number has a nonzero, high-order, hexadecimal fraction digit. If one or more high-order fraction digits are 0, the number is said to be unnormalized. The process of normalization consists of shifting the fraction left until the high-order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A fraction of 0 cannot be normalized and its associated characteristic therefore remains unchanged when normalization is called for.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result. This function is called postnormalization. For multiplication and division, the operands are normalized prior to the arithmetic process. This function is called prenormalization.

Most floating-point operations are performed only with normalization; a few are performed only without normalization. Addition may be specified either way.

When an operation is performed without normalization, high-order 0s in the result fraction are not eliminated. The result may or may not be normalized, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. Also, intermediate fraction results are shifted right when an overflow occurs, and the intermediate fraction result is truncated to the final result length after the shifting, if any.

Programming Note: Since normalization applies to hexadecimal digits, up to three high-order bits of a normalized fraction may be 0s.

3.4.4 Floating-Point Instruction Formats

Floating-point instructions use the RR and RX formats, as described in Section 3.1.2. In these formats, R1 designates a floating-point register. The contents of this register are called the first operand. The second operand location is defined differently for the two formats.

In the RR format, the R2 field specifies a floating-point register containing the second operand. The same register may be specified for the first and second operands. The register specified by the R1 and R2 fields should be 0, 2, 4, or 6. Otherwise, a specification exception is recognized, and a program interruption occurs.

In the RX format, the contents of the general register specified by X2 and B2 are added to the contents of the D2 field to form an address designating the location of the second operand. A value of zero in an X2 or B2 field indicates the absence of the corresponding address component.

The storage address of the second operand should be on a fullword boundary. Otherwise a specification exception is recognized, causing a program interruption.

Results replace the first operand, except for storing operations, where they replace the second operand. The contents of all other floating-point or general registers and storage locations that participate in the addressing or execution part of an operation remain unchanged.

The floating-point instructions are the only instructions that use the floating-point registers.

3.4.5 Decimal Floating-Point Instructions

Decimal floating-point instructions perform calculations on decimal data with a wide range of magnitudes.

Decimal Floating-Point Arithmetic

Decimal floating-point arithmetic combines certain features of packed decimal arithmetic and true (hexadecimal) floating-point arithmetic. Like packed decimal numbers, decimal floating-point numbers appear in BCD format rather than the hexadecimal format of true floating-point numbers. Like hexadecimal floating-point numbers, decimal floating-point numbers are represented by sign, characteristic, and mantissa values, and undergo arithmetic manipulations analogous to those for hexadecimal floating-point numbers. Therefore, decimal floating-point arithmetic can operate on numbers with a wide range of magnitudes and yield results scaled to preserve precision, without requiring conversion between decimal and hexadecimal representations.

The format of decimal floating-point numbers is shown in Figure 3-6.

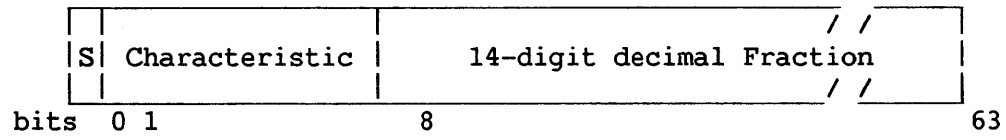


Figure 3-6. Decimal Floating-Point Number Format

A decimal floating-point number consists of a sign bit (S), a binary exponent (characteristic), and a decimal mantissa (fraction). The fraction consists of decimal digits (0-9) packed two to a byte, with the radix point of the fraction assumed to fall immediately to the left of the high-order fraction digit. The quantity expressed by this number is the signed product of the fraction and the number 10 raised to the power of the characteristic. The characteristic is expressed in excess-64 binary notation and ranges from -64 to +63.

Decimal floating-point arithmetic may use the four 8-byte floating-point registers for data manipulation. Decimal floating-point instructions provide both normalized RR and normalized RX formats for arithmetic operations -- that is, for addition (AQR and AQ), subtraction (SQR and SQ), multiplication (MQR and MQ), and division (DQR and DQ). Instructions in RX format for conversion between packed decimal and decimal floating-point numbers are CVP and CVQ. Load, store, and compare operations for decimal floating-point numbers employ the same instructions used for hexadecimal floating-point numbers.

Invalid digits cause data exceptions in all arithmetic and conversion instructions; data exceptions cause the instruction to be suppressed and leave the result unchanged. Invalid digits are not detected in LOAD, STORE, and COMPARE instructions.

3.5 LOGICAL INSTRUCTIONS

Logical information is handled as fixed- or variable-length data. It is subject to such operations as comparison, translation, editing, bit testing, and bit setting.

3.5.1 Fixed-Length Logical Data

When used as a fixed-length operand, logical information can consist of from one to four bytes and is processed in the general registers. Figure 3-7 shows the structure of fixed-length logical operands.

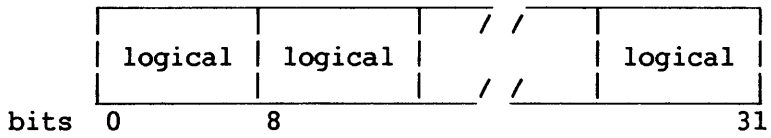


Figure 3-7. Fixed-Length Logical Operand (one to four bytes)

3.5.2 Variable-Length Logical Data

A large portion of logical information consists of alphabetic or numeric character codes, called alphanumeric data, and is used for communication with character-set-sensitive I/O devices. This information is in variable-field-length format and can be up to 256 bytes long. It is processed on a storage-to-storage basis, left to right, one byte at a time. Figure 3-8 shows the structure of variable-length logical operands.

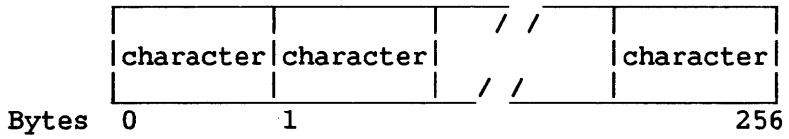


Figure 3-8. Variable-Length Logical Operand (up to 256 bytes)

The system can handle any 8-bit character set, although certain restrictions are assumed in decimal arithmetic and editing operations. However, all character-set-sensitive I/O equipment will assume the USA Standard Code for Information Interchange (USASCII) extended to eight bits, with the parity bit always set to 0 internally. In this manual the character set is referred to as USASCII-8 or simply, ASCII. The numbering convention for bit positions within a character or byte is as follows:

```
Bit positions  0 1 2 3 4 5 6 7
USASCII-8      8 7 6 5 4 3 2 1
```

Graphics are not defined for all 256 8-bit codes. When it is desirable to represent all possible bit patterns, a hexadecimal representation may be used instead of the 8-bit code. Hexadecimal representation uses one graphic for a 4-bit code, and therefore, two graphics for an 8-bit byte. The graphics 0-9 are used for codes 0000-1001; the graphics A-F are used for codes 1010-1111.

3.6 SUMMARY OF DATA FORMATS

Table 3-2 summarizes the data formats for the instructions described in Sections 3.1 through 3.4. The relative addresses for a series of numbers are given in order to illustrate boundary alignments of fixed-point and floating point data.

Table 3-2. Data Representation and Boundary Alignment

Data Type	Decimal Value	Hexadecimal Representation	Relative Address
Floating-Point	+4.3	4144CCCCCCCCCCCD	000000
"	-4.3	C144CCCCCCCCCCCD	000008
"	4.56E+5	456F540000000000	000010
"	4.56E-5	3C4C810D05CCF38E	000018
Decimal Floating-Point	+4.3	4143000000000000	000020
" "	-4.3	C143000000000000	000028
" "	4.56E+5	4645600000000000	000030
" "	4.56E-5	3D2FD0A823A01839	000038
Fixed-Point			
Binary fullword	+123	0000007B	000040
"	-123	FFFFFF85	000044
Binary halfword	123	007B	000048
Decimal			
Packed	+123	123F	00004A
"	-123	123D	00004C
Zoned	+123	3132F3	00004E
"	-123	3132D3	000051
External	123	313233	000054
"	+123	2B313233	000057
"	-123	2D313233	00005B
"	123+	3132332B	00005F
"	123-	3132332D	000063
Logical	'DATA'	44415441	000067

3.7 LINKED LIST INSTRUCTIONS

The instructions ENQ, ENSK, DEQ, and DESK handle lists of blocks connected by pointers. Two kinds of linked lists are supported: last-in first-out (LIFO) lists and first-in first-out (FIFO) lists. The instructions provide the means to add to and delete from the lists, and to determine whether the lists are empty.

3.7.1 Structure of LIFO Lists

The LIFO header consists of an aligned word containing either a null pointer (0s) or the address of the first block in the list. This address, or pointer, is in the low-order three bytes of the word. Each block in the list also contains either a null pointer or the address of the start of the next block in the list. Figure 3-9 is a diagram of such a list. The pointers in the blocks are all at a displacement into the block determined displacement field of the ENSK or DESK instruction.

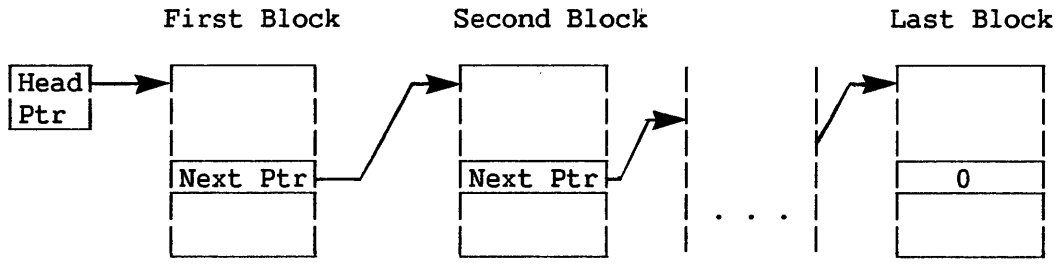


Figure 3-9. LIFO List

3.7.2 Structure of FIFO Lists

The FIFO list, pictured in Figure 3-10, consists of head and tail pointers in consecutive words, doubleword aligned, and the chain of blocks addressed by the head and tail pointers. If the list is empty, the head and tail pointers are null (0). If the list is not empty, the head pointer addresses the start of the first block in the list, and the tail pointer addresses the start of the last block. If there is only one block, the head and tail pointers are the same. In the blocks the pointers will be exactly the same as for the LIFO list.

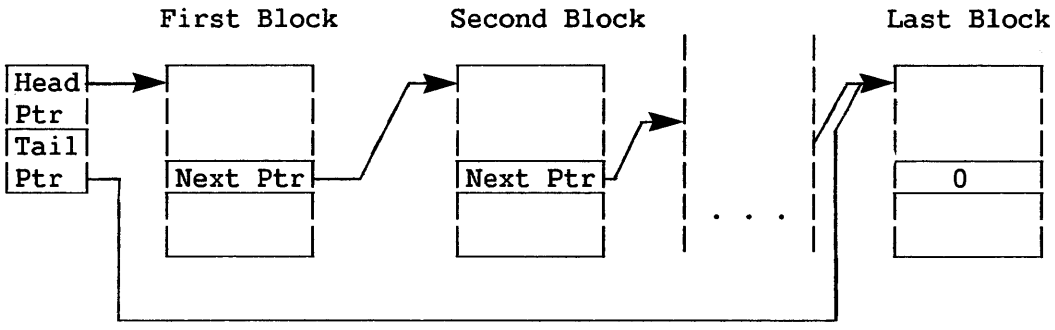


Figure 3-10. FIFO List

3.8 SEMAPHORE MANIPULATION INSTRUCTIONS

The Decrement and Inspect Semaphore (DSEM) and Increment and Inspect Semaphore (ISEM) instructions operate on a unique doubleword data type, the semaphore, consisting of linked list head and tail pointers and a 1-byte count field. The semaphore data type is illustrated in Figure 3-11. The semaphore must be aligned on a doubleword boundary. These pointers contain addresses of a FIFO list, and are manipulated exactly as for the FIFO list instructions.

These instructions may be used to control sharing of a system resource (e.g., processor, memory, or I/O devices). The DSEM instruction is issued when a unit of the resource is to be requested, and the ISEM instruction is issued when a unit of the resource is to be released. The conditional branching effected, which is contingent on the contents of the count field, allows the program to prevent the allocation of more units of the resource than are specified by the initial positive value of this field. For details of instruction execution, refer to the particular instruction descriptions in Chapter 8.

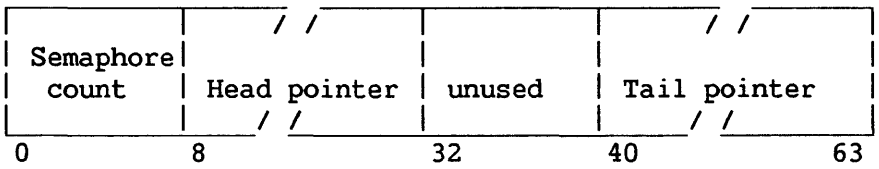


Figure 3-11. Semaphore

3.9 STACK-ORIENTED INSTRUCTIONS

The stack-oriented feature consists of the BALS, BCS, SVC, SVCX, JSCI, RTC, PUSH, PUSHM, PUSHC, PUSHN, POP, POPH, POPM, POPC, and POPN instructions, which operate on a pushdown list in descending memory locations. This list is addressed through two address words (stack pointer and stack limit word, in that order) that may be either in General Register 15 and Control Register 2 (which constitute the system stack vector) or in two consecutive general registers (the user stack vector). If the S1 (or S2 for BCS) field of one of these instructions is 0, the system stack vector is used. Otherwise the S1 (or S2 for BCS) field addresses the general register containing the stack limit. The previous register will be the stack pointer.

The stack limit word addresses the lowest byte location into which the stack may extend as it grows into successively lower addressed locations. The stack pointer addresses the current stack top, that is, the lowest byte location that contains stacked information. Note that the stack pointer of the system stack vector is in General Register 15. The value in the stack pointer decreases as items are placed on a stack.

Items, including character strings, are placed on stacks in word-aligned locations. The stack pointer must address a fullword boundary (that is, have two low-order zero bits) before any stack-oriented instruction is processed, or a specification error will result and the instruction will be suppressed. Thus, registers may dependably be loaded from stacks by L, LH, and LM instructions. They may also be loaded by POP, POPM, and ICM instructions.

When bytes are placed on a stack by the PUSHN or PUSHC instructions, sufficient bytes are skipped (unmodified) before pushing any bytes so that the stack pointer addresses a fullword boundary when the instruction is completed. Thus zero, one, two, or three bytes may be skipped. When bytes are removed from the stack by the POPN or POPC instructions, sufficient additional bytes are popped and discarded (as for POPN) so that the stack pointer addresses a fullword boundary when the instruction is completed.

The previous contents of the high-order byte of words in the stack vector are irrelevant to all stack-oriented instructions. The high-order byte of the stack pointer will be set to zero whenever this word is modified by one of these instructions. The stack limit word is unchanged by these instructions.

3.10 STACK SWITCHING

In a task's virtual address space, a system stack is maintained for each process level at which the task may execute. The separation of stack areas reduces the possibility of programs at different process levels overwriting each other's modifiable data.

When one routine calls another that executes at a higher process level, the system stack associated with the called routine is activated; that is, the system stack vector is updated to describe the status of the called routine's stack. Upon return of control to the caller, the caller's stack is reactivated.

The activation of a new stack is called a stack switch. Stack switching is implemented through a data structure called a Stack Header Block (SHB). There is a SHB for each process level and stack associated with that level. Entries in the SHB point to the top of stack and the stack limit. Another SHB field points to the most recently built JSCI save area. (The JSCI save area is described in Section 4.6.1.) Control Register 8 points to the active SHB (that is, the SHB associated with the currently active stack). The SHB is illustrated in Figure 3-12.

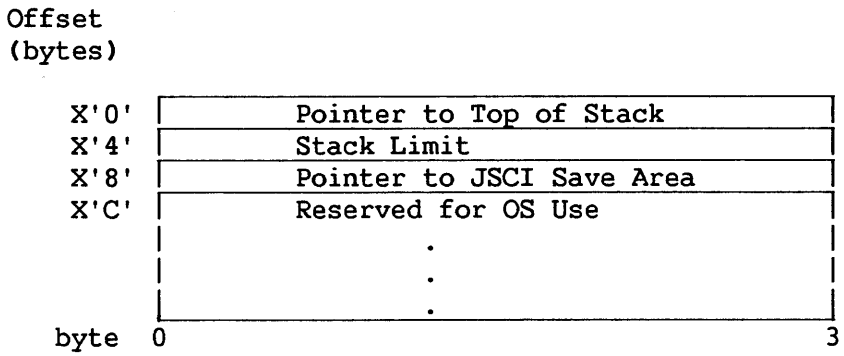


Figure 3-12. Format of Stack Header Block (SHB)

The length of each SHB entry is 4 bytes, of which the first is zero.

Stack Header Blocks are are addressed through the Stack Header Block table (SHBT), which in turn is pointed to by Control Register 9. The table consists of eight 4-byte entries. The rightmost three bytes of each entry hold the address of the SHB associated with a process level. The entries address the SHBs for Process Levels 0-7 in ascending order of process level.

The SHB associated with a particular process level can be located by indexing into the table by the level number. The process level of a called routine can be found in the Linkage table, described in Section 4.6.2. The process level of the calling routine is recorded in the JSCI save area.

When a stack switch involves a transition to a higher level, the following occurs: the current value of the stack vector (GR15 and CR2) and the JSCI save area back chain (CR1) are saved in the SHB pointed to by CR8. The address of the new SHB, associated with the called routine, is extracted from Stack Header Block table and loaded into CR8. GR15 and CR2 are loaded from the new SHB pointed to by CR8. The system vectors now describe the stack used by the called routine.

When the RTC or RPC instruction returns control to the caller, another stack switch takes place. GR15 and CR2 are saved in the called routine's SHB, which is still pointed to by CR8. CR1 points to the save area on that stack, where the caller's process level has been stored. CR9 points to the Stack Header Block table. The caller's process level provides an index to the address of the caller's SHB in the SHBT. This address is loaded into CR8. GR15 and CR2, comprising the stack vector, are loaded from the caller's SHB, so that they now describe the caller's stack.

For further information on stack switching, refer to the descriptions of the JSCI and RTC instructions in Chapter 8.

CHAPTER 4
INSTRUCTION EXECUTION

4.1 PROGRAM CONTROL WORD

The Program Control Word (PCW) is eight bytes long and contains the information required for proper program execution. It includes status and control information, interruption codes, and the instruction address. Uses of the PCW are explained in Chapter 5. In general, the PCW is used to control instruction sequencing and to indicate the status of the system in relation to the program currently being executed.

To execute a sequence of instructions, the CP takes the address of an instruction from the PCW. It executes that instruction and increments the PCW's instruction address by the length of the instruction. It then takes the new instruction address from the PCW. The process continues until an interruption occurs or a branching instruction is executed.

The active or controlling PCW is called the current PCW. Through storage of the current PCW, the status of the CP can be preserved for subsequent inspection. Through loading of a new PCW or part of a PCW, the state of the CP can be changed. The PCW is made up of a 1-byte interruption code (discussed in Chapter 5), a 3-byte instruction address, a 2-byte status field, a 1-byte program mask field, and a 1-byte field whose three low order bits indicate process level. The PCW for a program can be inspected through Debug mode or by doing a program dump. Figure 4-1 shows the PCW format.

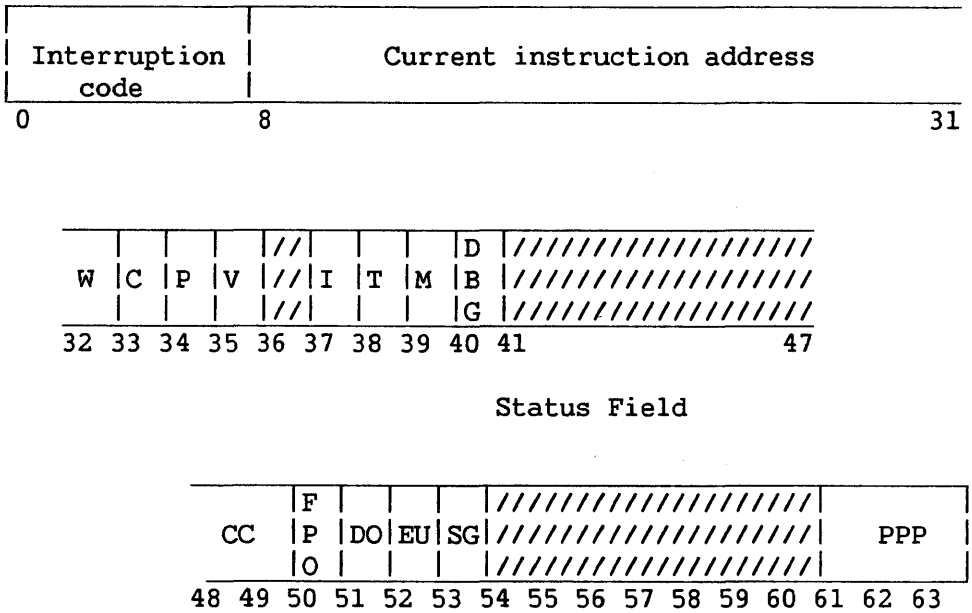


Figure 4-1. PCW Format

Table 4-1 gives a more detailed explanation of the function of each bit in the PCW.

Table 4-1. PCW Bits

PCW Bits	Mnemonic	Function
0-7		Interruption code
8-31		Current instruction address
System Mask Field		
32	W	Wait state 0 = Operating state 1 = Wait state
33	C	Control mode 0 = Normal operating mode 1 = Control mode

(continued)

Table 4-1. PCW Bits (continued)

PCW Bits	Mnemonic	Function
34	P	Memory protection violation and privileged instruction trap 0 = Do not trap on attempt to access protected memory or execute privileged instruction 1 = Trap on attempt to access protected memory or execute privileged instruction
35	V	Virtual machine 0 = Native machine mode 1 = Virtual machine mode
36		Reserved
37	I	I/O interruption mask 0 = I/O interruptions disabled 1 = I/O interruptions enabled
38	T	Clock interruption mask 0 = Clock interruptions disabled 1 = Clock interruptions enabled
39	M	Machine check interruption mask 0 = Machine check interruptions disabled 1 = Machine check interruptions enabled
40	DBG	Debug control bit 0 = Debug traps disabled 1 = Debug traps enabled
41-47		Reserved
48-49	CC	Condition code
Program Mask Field		
50	FPO	Fixed-Point overflow mask 0 = Do not interrupt on overflow 1 = Overflow will cause interruption

(continued)

Table 4-1. PCW Bits (continued)

PCW Bits	Mnemonic	Function
51	DO	Decimal overflow mask 0 = Do not interrupt on overflow 1 = Overflow will cause interruption
52	EU	Exponent underflow mask (floating-point instructions) 0 = Do not interrupt on underflow 1 = Underflow will cause interruption
53	SG	Significance mask (floating-point add or subtract instructions) 0 = Do not interrupt on zero intermediate sum 1 = Zero intermediate sum will cause interruption
54-60		Reserved
61-63		Process level

4.1.1 Wait State

When the Wait state bit = 1, the CP does not execute machine instructions. Three external events cause the CP to leave wait-state: an enabled clock interrupt, an enabled I/O interrupt, and pressing of the Control mode button.

When the CP is in Wait state and a clock or I/O interrupt becomes active, the CP, taking the interrupt, loads the new PCW, which typically brings the CP to normal operating state. When Control mode entry is detected, the CP enters that state while preserving the current PCW value.

The CP does not turn off the Wait state bit when leaving Wait state. That bit is typically turned off by the loading of a new PCW or, upon exit from Control mode, by the reloading of the current PCW.

4.1.2 Condition Codes

The condition code is a 2-bit field in the PCW that can be tested by many of the instructions. Once the code is set, it is changed only by certain instructions, such as ADD, COMPARE, SET PROGRAM MASK, and LOAD PCW. The meanings of the condition codes for each instruction are listed under that instruction in Chapter 8.

4.1.3 Process Levels

The process level is a 3-bit field in the PCW that indicates the current level of execution. Levels of execution range from 0 to 7.

Only programs running at Level 7 may execute privileged instructions. A comparison of process level to access levels determines a program's access rights to memory.

The access levels for a region of memory are defined in the Region Table entry describing that region. Each entry includes a 3-bit read access field and a 3-bit write access field. (The Region Table and its entries are fully described in Section 4.3.5.) The values of these fields signify the minimum process level from which the associated region can be read or written to. Thus, to read a region whose read access level is n, a program must be executing at a process level equal to or greater than n. The value of the write access field must be that of the read access field or 7.

4.2 ADDRESSING

For addressing purposes, operands can be grouped in three classes: explicitly addressed operands in main memory, immediate operands placed as part of the instruction stream in main memory, and operands located in registers.

To permit the ready relocation of program segments and flexible specification of input, output, and working areas, most instructions referring to main memory can employ a full address. A full address consists of a base address, an index, and a displacement. For instructions in the RL and RRL format, the base address is implicitly the current instruction address (relative addressing). Instructions in other formats explicitly specify a base address in their B field (base-displacement addressing).

To address the first 4096 bytes of memory, an instruction can specify an address by the displacement element alone (direct addressing).

4.2.1 Base-Displacement Address Generation

Base-displacement addresses are generated from the following three binary numbers:

- Base Address (B) is a 24-bit number contained in a general register specified by the program in the B field of the instruction. The B field is included in every address specification. The base address can be used for static relocation of programs and data. In array calculations it can specify the location of an array, and in record processing it can identify the record. The base address provides for addressing all of main memory. The base address may also be used for indexing purposes.

- Index (X) is a 24-bit number contained in a general register specified by the program in the X field of the instruction. It is included only in the address specified in the RX instruction format. The RX format instructions permit double indexing; that is, the index can be used to provide the address of an element within an array.

- Displacement (D) or offset is a 12-bit number contained in the instruction format. It is included in every address computation. The displacement provides for relative addressing of up to 4095 bytes beyond the element or base address. In array calculations the displacement can specify one of many items associated with an element. In processing records, the displacement can identify items within a record.

In forming the address, the base address and index are treated as unsigned 24-bit binary integers. The displacement is similarly treated as an unsigned 12-bit binary integer. The three are added as 24-bit binary numbers, ignoring overflow. Since every address includes a base, the sum is always 24 bits long.

The program may show a value of zero in the base address, index, or displacement field. A zero indicates the absence of the corresponding address component. A base or index of zero implies that a value of zero is to be used in forming the address, and does not refer to the contents of General Register 0. Thus, the use of Register 0 as a base register makes a program unrelocatable. A displacement of zero has no special significance. Initialization, modification, and testing of base addresses and indexes can be carried out by fixed-point instructions, or by BRANCH AND LINK, BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL instructions.

4.2.2 Relative Address Generation

For instruction formats (RL and RRL), a base register is unnecessary. The current instruction address is an implied base address, and a relative offset is added to it to form the effective address. Use of this format is limited to five branch instructions, RLA, and RPUSHA.

The address used to refer to main memory is generated from the following three binary numbers:

- Current instruction address is the implied base address. So, for example, if both X and L values (see below) are zero, then the instruction branches to itself.

- Index (X), if specified in the instruction, is a 24-bit number contained in a general register specified by the program in the X field of the instruction.

- Relative Offset (L) is extended from the number of bits in the instruction to a 24-bit number.

In forming the address, these three numbers are added as unsigned 24-bit binary integers, ignoring overflow.

4.2.3 Direct Address Generation

Addresses 0-4095 can be generated without a base address or index. This property is important when the PCW and general register contents must be preserved and restored during program switching. These addresses further include all reserved addresses used by the system for fixed purposes, such as old PCWs, new PCWs, and IOSW and IOCT locations.

4.3 ADDRESS TRANSLATION

All VS systems provide many users simultaneously with a virtual address space for instructions and data that is larger than the amount of memory physically available to the system; in fact, VS is an acronym for virtual storage. Most of this virtual address space is located on disk.

Because instructions and data must be present in main memory (i.e., physical memory) while being processed, they are copied from disk into main memory as needed. The process of copying information from virtual memory into main memory is called paging. Paging is accomplished in units of 2 KB, or one page, by a part of the operating system called the Pager.

Before a program instruction can be executed, a conversion must be performed on the virtual addresses specified within it. The process of converting virtual addresses into physical main memory addresses is called address translation. A combination of hardware and operating system action translates each virtual address as it is encountered during program execution.

Because the programs of many users exist in physical memory simultaneously and only one of these can be processed at a time, a means of working for short "time slices" successively on different programs is implemented in the operating system. Time slices give the effect of simultaneous action on a number of programs.

The process of preparing conditions for the CP to work first on one program, then another, is called context switching. Context switching also is accomplished by a combination of hardware and operating system action. As a part of context switching, the translation of one user's virtual address space is discontinued and that of another user's is begun.

4.3.1 Physical/Virtual Address Space

Main memory for all VS machines consists of byte-addressable random access memory (RAM). Located on semiconductor chips in the CP cabinet, main memory is divided logically into page frames of 2 KB, each aligned on a 2-KB boundary and containing exactly one page of information.

Main memory addresses consist of a 15-bit page frame number and an 11-bit byte index to locations within the page, as illustrated in Figure 4-2.

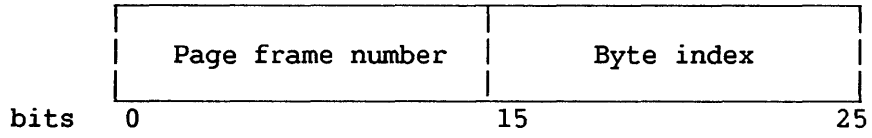


Figure 4-2. Physical Address Format

The range of main memory addresses depends upon the amount of physical memory configured into the installation, and currently varies from 512 KB to 64 MB (i.e., 2**26 = 64M) with different processors of the VS family.

Virtual memory, located in disk storage, is divided logically into pages and regions. Virtual memory addresses consist of a 13-bit virtual page index and an 11-bit byte index to locations within the page, as illustrated in Figure 4-3. The 24-bit virtual address allows for up to 16 MB of addressable storage (i.e., 2**24 = 16M). Virtual pages are 2 KB in size, beginning on a 2-KB boundary; physically, each page occupies one sector of a disk platter. Regions are blocks of pages, variable in size, and beginning on a 2-KB boundary. The VS architecture supports the segmentation of a task's address space into 512 regions; a smaller number, however, is actually implemented by the operating system. Pages of virtual memory are copied as needed into available page frames of main memory, as discussed in Section 4.3.2.

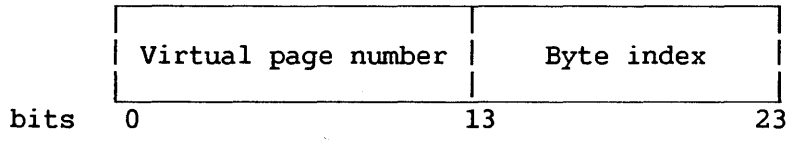


Figure 4-3. Virtual Address Format

4.3.2 Main Memory Page Tables

A task's page tables form an essential part of the address translation mechanism. A page table is a section of main memory that defines the mapping of virtual address space to main memory page frames. There is a page table for each region of a task's address space. The format of page table entries is illustrated in Figure 4-4.

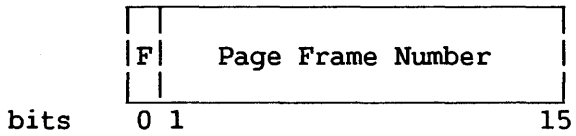


Figure 4-4. Main Memory Page Table Entry Format

Each halfword entry of the page table corresponds to a virtual page address. If the page exists in main memory, the fault bit (F) is equal to 0 and Bits 1-15 are its page frame number. If not, F=1 and the rest of the entry is not examined.

To translate a virtual address, the CP first indexes into the page table to the appropriate page table entry. The correct index, or offset, is obtained as follows: the first entry of a page table maps the lowest page number in a region to physical memory. The lowest page number in a region is recorded in a Region Table, as explained in Section 4.3.5. The residual difference between the 11-bit page number of a virtual address and the lowest page number in a region supplies the index to the page table entry corresponding to that page number.

If the fault bit of that entry is off (0), the CP concatenates the 15-bit page frame number number in the entry with the 11-bit byte index of the virtual address to form a 24-bit physical address. The CP then uses this physical address to access the data in memory. If the fault bit is on (1), the CP signals the operating system (i.e., the pager) that a page fault has occurred. The pager copies the virtual page from disk into an available page frame and records the number of the selected page frame in the page table. The task can then resume execution.

4.3.3 Local Page Table

To speed up address translation, a subset of the currently executing task's page tables is held in local CP memory and is checked first during translations. Most instances of address translation are accomplished by reference to this local page table.

Translation RAM (T-RAM)

The local page table of VS15, VS65, and VS100 systems is called Translation RAM (T-RAM). There are as many 2-byte entries in T-RAM as pages in the user's virtual address space.

At the start of each user's time slice, the fault bits of T-RAM entries are set to 1, indicating that they do not hold valid page frame numbers. Then, as pages are referenced by the task, page frame numbers found in main memory page tables are also recorded in the T-RAM, and subsequent references to these pages during the same time slice are satisfied from the T-RAM in 720 nanoseconds rather than the 20 microseconds (approximate times estimated for the VS15) required for a main memory page table access.

T-RAM entries include memory protection fields that regulate read and write access to the corresponding page frame. The format of these fields varies between VS systems, as explained in Section 4.3.7.

Translation Buffer (T-BUF)

The local page table of the VS300 system is called a translation buffer (T-BUF). T-BUF, like T-RAM, holds a subset of the currently executing task's page tables, and is checked first during translations. T-BUF entries include a physical page frame number, fault bit, monitor bit, and protection fields.

There are 1024 entries in the buffer. One entry is associated with a set of eight virtual page numbers in the user's address space. These numbers increase by increments of 1K, so that Bits 3 through 12 of the virtual page numbers in a set have the same value.

At the start of each user's time slice, the fault bits of T-BUF entries are set to 1, indicating that they do not hold valid page frame numbers. Then, as pages are referenced by the task, page frame numbers found in main memory page tables are recorded in T-BUF.

To translate a virtual address, the CP first locates the T-BUF entry associated with the virtual page number of the address, using Bits 3 through 12 as an index. If the entry's fault bit is on, the appropriate physical page frame number is copied from a main memory page table to the T-BUF entry. The most significant three bits of the virtual address being translated are placed in a tag store area in local memory. This area consists of 1K entries, each paired with a T-BUF entry. Having indexed to a valid entry, the CP compares the three most significant bits of the virtual address being translated with the contents of the tag store entry paired with the T-BUF entry. This comparison determines whether the physical page frame number in the T-BUF entry maps to the virtual page number of the address being translated or to one of the seven other virtual pages in the set. If the comparison results in a match, a full physical address is generated by concatenation, as described in Section 4.3.2. Otherwise, a page table in main memory is referenced for the correct physical page frame number.

4.3.4 Segment Control Registers (SCRs)

Segment Control Registers (SCRs), along with Region Tables, are used to locate the page table applicable to translating a particular virtual address. At the beginning of a task's time slice, its four SCRs are loaded with information that typically takes the form shown in Figures 4-5 and 4-6.

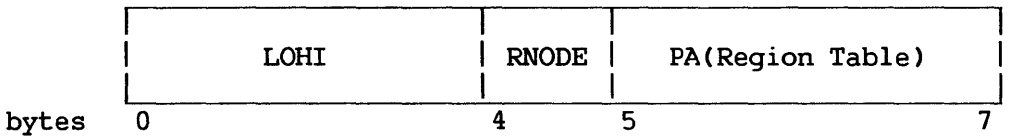


Figure 4-5. Format of SCR

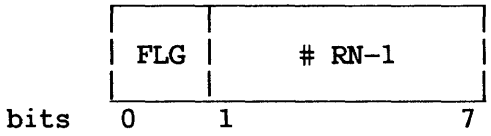


Figure 4-6. Byte 4 of SCR

The first word of each SCR is a LOHI word, which indicates the virtual address range covered by the corresponding Region Table. The first 16 bits of this word indicate the starting virtual page number; the other 16 bits indicate the ending virtual page number plus 1. On a T-RAM fault, the 13-bit virtual page number of the faulted address is compared with each LOHI word until a match is found.

Byte 4 of the SCR, shown in Figure 4-6, contains a 1-bit flag and a 7-bit field that specifies the number of regions nodes minus 1 in the associated Region Table. VS architecture supports up to 128 Region Nodes per table; the operating system, however, may support fewer. The value of the flag is typically 0, which indicates that the format of Bytes 4-7 is that shown in Figure 4-5. A value of 1 indicates the alternate format shown in Figure 4-10, which is used for translating a virtual address without reference to a page table.

Bytes 5 to 7 contain the physical address of the associated Region Table. The use of a physical address avoids possible recursion of translation (i.e., translating a second virtual address in order to locate the Region Table used to translate the target virtual address).

The contents of SCRs are loaded and stored through the privileged LSCTL and STSCTL instructions, described in Chapter 8.

4.3.5 Region Table

A Region Table contains one or more 16-byte entries called Region Nodes; each entry describes a region and points to the page table for that region. Figure 4-7 illustrates the format of the first 8 bytes of a Region Node; the remaining bytes are reserved for use by the operating system.

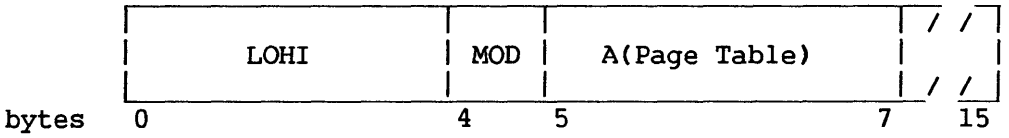


Figure 4-7. Format of a Region Node.

LOHI Word

The LOHI word in the first four bytes of each entry is identical in format to the LOHI word of the Segment Control Register (see Section 4.3.4). On a T-RAM or T-BUF fault, the 13-bit virtual page number of the faulted address is compared with each LOHI word; when a match is found, the page table has been located. The residual difference between the virtual page number and LO page number is used to index into the page table.

MOD Byte

The format of the fifth, or MOD, byte is shown in Figure 4-8.

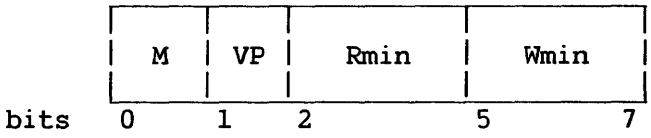


Figure 4-8. Format of MOD Byte

Bit 0, the M bit, indicates whether monitoring is in effect for the region. When monitoring is in effect (i.e., when M=1), it records each page table entry loaded into the T-RAM or T-BUF from the main memory page table pointed to by Bytes 5-7. For a discussion of monitoring, refer to Section 4.3.7. Bit 1, the VP bit, indicates whether the Region Table address in Bytes 5-7 is physical or virtual. A value of 1 for the VP bit indicates a physical address.

The 3-bit Rmin and Wmin fields specify the minimum read and write access levels for all pages in the region. During virtual address translation, the values of Rmin and Wmin fields are compared with the current process level (i.e., the level of the process that has specified in an instruction the virtual address being translated). A process level equal to or greater than the Rmin or Wmin values is required, respectively, for read and write access to memory locations mapped to the region. The possible values of the Rmin and Wmin fields are described in Section 4.1.3.

4.3.6 Summary of Address Translation

To translate a virtual address, the CP first locates the corresponding entry in the T-RAM or T-BUF, using part of the address as an index. If the page into which the virtual address falls has already been referenced by the task, the entry contains a valid page frame number. Using this number, the CP translates the virtual address by concatenation, as described in Section 4.3.2.

If the T-RAM or T-BUF entry does not hold a valid page frame number, address translation proceeds as shown in Figure 4-9. The page index of the virtual address is matched to the LOHI range of the task's Segment Control Registers (SCRs). (Figure 4-9 shows only two of the task's four SCRs.) The SCR whose LOHI range includes the page index points to the appropriate Region Table. The sample Region Table in Figure 4-9 consists of two Region Nodes. The node whose LOHI range includes the page index points to the appropriate page table. The residual difference between the page index of the virtual address and the LO page number of the Region Node supplies an index to the correct page table entry. The fault bit of the entry is checked to determine whether the page indicated by the virtual address is currently in memory. If the fault bit = 0, the page frame number of the page table entry is concatenated with the byte index of the virtual address to form a physical address. Otherwise, the paging task is called to load the page in memory.

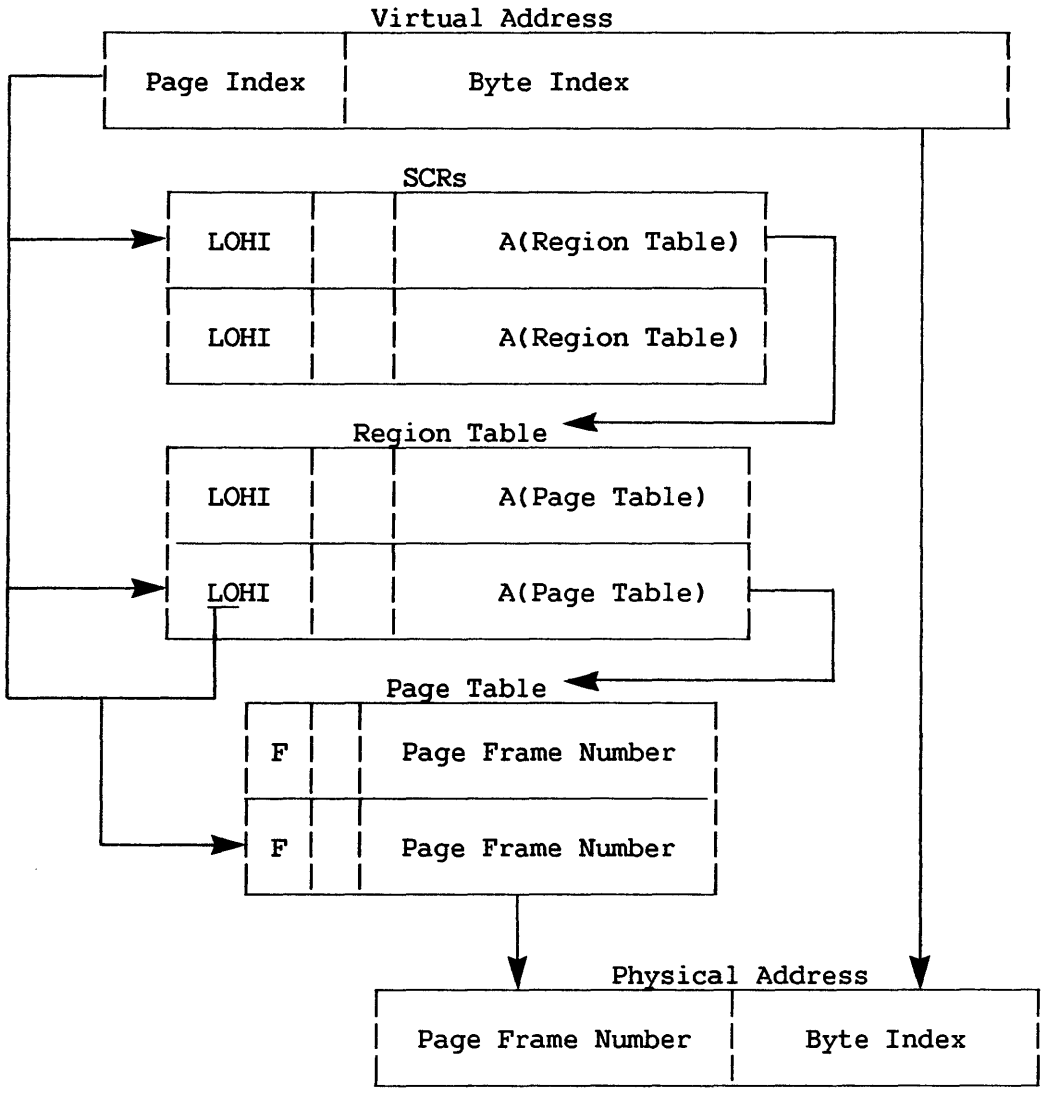


Figure 4-9. Virtual-to-Physical Address Translation

4.3.7 T-RAM Monitor Area

The Monitor Area of memory is used for recording the virtual page numbers stored in T-RAM entries. The Monitor Area is referred to during the clearing of T-RAM entries, which takes place at a change from higher to lower process level during a task's time slice or at the end of a task's time slice. For the VS65 system, the Monitor Area is located in main memory, at an address determined by the operating system and passed to the CP; for the VS15 and VS100 systems, the Monitor Area is in local CP memory.

Monitoring is enabled for a region of a user's virtual memory if the M-bit is set in the Region Node for the region; refer to Figures 4-7 and 4-8 for an illustration of Region Node format. During the successful servicing of a T-RAM fault for a virtual page in a region for which monitoring is enabled, the virtual page address is recorded in the next available Monitor Area location. At a downward change of process level or at the end of a user's time slice, only those T-RAM entries identified by the monitor are cleared (i.e., their high order bit is set to 1), rather than the entire T-RAM. Because only a fraction of all the T-RAM entries is likely to be loaded during a given time slice, using the monitor greatly reduces the number of T-RAM entries to be cleared.

There is no Monitor Area on the VS300 system. Instead, each T-BUF entry includes a monitor bit. When monitoring is enabled for a region (through the M-bit of that region's node) and a T-BUF entry is loaded from that region's page table, the T-BUF monitor bit is set to 0. At the end of a task's time slice, the CP can selectively clear those T-BUF entries whose monitor bits are set to 0.

As explained in the remainder of this section, the memory protection fields of T-RAM entries vary between classes of VS systems. Accordingly, there are variations between VS systems in the composition of the Monitor Area and the clearing of the T-RAM.

VS15 and VS100 Monitor Area

In these classes of VS systems, the T-RAM entry includes a 2-bit protection field supporting read/write access (bit values = 00) and read access only (bit values = 01). These bits are set during address translation after a comparison of the current process level (indicated in the current PCW) with the read minimum (Rmin) and write minimum (Wmin) fields in the appropriate Region Node. Rmin and Wmin values range from 0 to 7. When a page is first referenced by a process running at Level 7, the protection bits in the appropriate T-RAM entry are always set to 00, because the process level is equal to or greater than any possible values in the Rmin and Wmin Region Node fields. A downward change of process level renders the value of T-RAM protection bits inapplicable. Thus, to maintain memory protection, T-RAM entries are cleared on downward changes of process level, as well as at the end of a task's time slice, as explained in the next paragraph.

The Monitor Area consists of an executive list and a user list. The executive list records virtual page numbers that are mapped to physical memory when a task runs at a process level higher than zero. The user lists records virtual page numbers mapped at Process Level 0. When execution of the RTC or RPC instruction lowers the process level to zero, the executive list is used to clear those T-RAM entries added at the higher process level. In this way, the differentiation of memory access rights according to process level is efficiently enforced. On a downward change of process level to a nonzero level, all entries recorded in the executive list are cleared. At the end of a task's time slice, the RRCB instruction clears the T-RAM using both executive and user monitor lists.

The VS15 architecture supports a Monitor Area of 64 entries, 32 entries per list. The VS100 architecture supports a Monitor Area of 128 entries, 64 entries per list.

VS65 Monitor Area

On this VS system, the T-RAM entry memory protection field includes a process level subfield (3 bits) and a write protect bit. The process level field defines the minimum process level for read access. When the write protect bit is off, the minimum process level for write access is equal to that for read access; when the write protect bit is on, only processes running at Level 7 or in privileged state (PCW Bit 34 = 0) have write access. Because of this protection scheme, it is not necessary to clear T-RAM entries on downward transitions of process level. Accordingly, the Monitor Area is a single list.

4.3.8 Reference and Change Table

The reference and change table (RCT) makes possible the efficient replacement of old memory pages with new pages read in from disk. The RCT is an area of local CP memory containing an entry of two bits for each page of main memory. When some location in a page frame is referenced by a user program, the reference bit for the page frame is set to 1; when the location is also modified, the change bit is also set to 1. The operating system uses the reference and change bits along with an aging count in deciding which virtual pages to overwrite with new ones during paging operations.

4.3.9 Alternate SCR Format

When page tables are involved in address translation, they are located through Segment Control Registers, as explained in Section 4.3.6. Address translation, however, can also be accomplished without page tables, as when consecutive pages are fixed in consecutive page frames. When page tables are dispensed with, SCRs have a function and format different from that previously described. A value of 1 for the SCR flag bit indicates the alternate format illustrated by Figure 4-10.

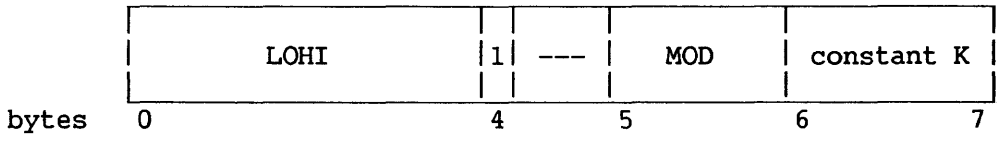


Figure 4-10. Alternate Format of SCR

The LOHI word in the alternate SCR format specifies the range of virtual addresses in a region. The page frame number corresponding to the page number in a virtual address equals the residual difference between that page number and 16-bit LO value in the SCR plus the constant K. That is, PFN = (VPN-LO)+K. The MOD byte is the same in format as the MOD byte of the Region Node, illustrated in Figure 4-8.

4.4 SEQUENTIAL INSTRUCTION EXECUTION

Normally, the operation of the CP is controlled by instructions taken in sequence. An instruction is fetched from a location specified by the instruction address in the current PCW. The instruction address is then increased by the number of bytes in the fetched instruction to address the next instruction in sequence. The instruction is then executed and the same steps are repeated using the new value of the instruction address. A change from sequential operation may be caused by branching, status switching, interruptions, or manual intervention.

4.5 BRANCHING

The normal sequential execution of instructions is changed when reference is made to a subroutine, when a 2-way choice is encountered, or when a section of coding, such as a loop, is to be repeated. All these tasks can be accomplished with branching instructions. Provision is made for subroutine linkage, permitting not only the introduction of a new instruction address but also the preservation of the return address.

Decision-making is generally and symmetrically provided by the BRANCH ON CONDITION instruction. This instruction inspects a 2-bit condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. Each of these operations can set the code to any one of four states, and the conditional branch can specify any selection of these four states as the criterion for branching. For example, the condition code reflects such conditions as nonzero; first operand high, equal, or low; overflow; I/O device busy; zero; etc. Once set, the condition code remains unchanged until modified by an instruction that sets it differently.

Loop control can be performed by the conditional branch when it tests the outcome of address arithmetic and counting operations. For some especially frequent combinations of arithmetic and tests, the instructions BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL are provided. These branches are specialized to increase performance for these tasks.

4.5.1 Instruction Formats

Branching instructions use the RR, RX, RS, RL, and RRL formats. In these formats R1 specifies the address of a general register. In BRANCH ON CONDITION, a mask field (M1) identifies the bit values of the condition code. The branch address is defined differently for the three formats.

In the RR format, the R2 field specifies the address of a general register containing the branch address, except when R2 is zero, which indicates no branching. The same register may be specified by R1 and R2.

In the RX format, the contents of the general registers specified by the X2 and B2 fields are added to the D2 field to form the branch address.

In the RS format, the contents of the general register specified by the B2 field are added to the contents of the D2 field to form the branch address. The R3 field in this format specifies the location of the second operand and implies the location of the third operand. The first operand is specified by the R1 field.

Programming Note: The third operand location is always odd. Thus, in instructions such as BXLE and BXH, if the R3 field specifies an even register, the third operand is obtained from the next higher addressed register. If the R3 field specifies an odd register, the third operand location coincides with the second operand location.

In the RL format, the current instruction address is added to the L2 field to form the branch address.

In the RRL format, the current instruction address is added to the X2 and L2 fields to form the branch address.

A zero in a B2 or X2 field indicates the absence of the corresponding address component.

A branching instruction can specify the same general register for both address modification and operand location. The order in which the contents of the general registers are used for the different parts of an operation is as follows:

1. Address computation
2. Arithmetic or link information storage.

Results are placed in the general register specified by R1. Except for the storing of the final results, the contents of all general registers and memory locations participating in the addressing or execution part of an operation remain unchanged.

Programming Note: In several instructions the branch address may be specified in two ways: In the RX format, the branch address is the address specified by X1, B2, and D2; in the RR format, the branch address is in the register specified by R2.

4.6 JSCI INSTRUCTION

4.6.1 JSCI Save Area

Branching to the entry point of a subroutine is accomplished by the JSCI instruction (described in Chapter 8). Before the branch is taken, the JSCI instruction pushes the context of the calling routine onto a stack. The information pushed on the stack consists of the program mask byte and the updated instruction address of the current PCW (which serves as the return address), the caller's process level, a back chain to any prior JSCI save area, and the general register contents. The stack area containing this information, called the JSCI save area, is illustrated in Figure 4-11.

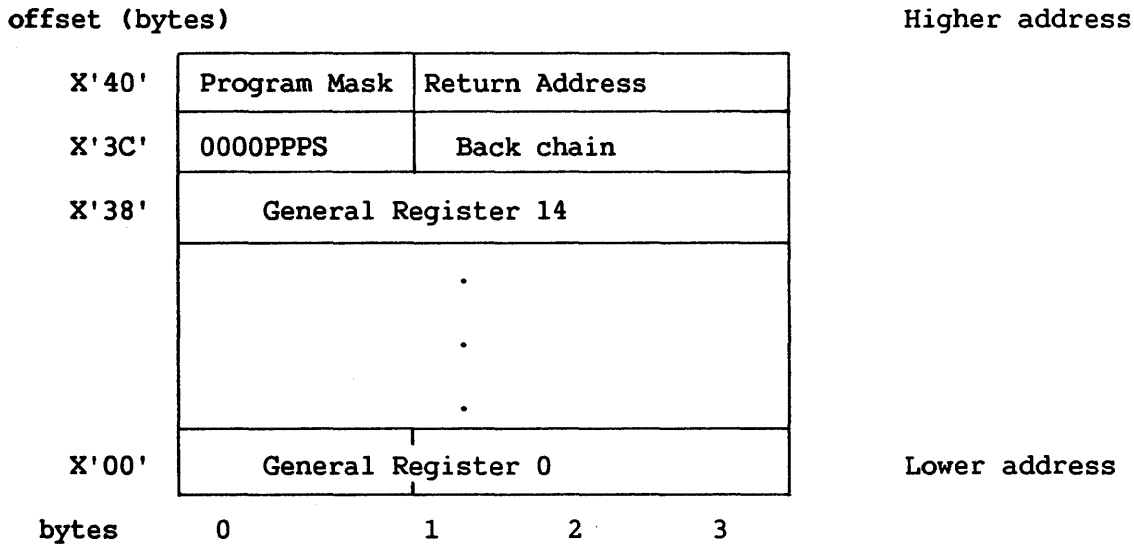


Figure 4-11. JSCI Save Area

PPP (at offset X'3C') is a 3-bit field indicating the caller's process level. When the caller's process level is equal to the process level of the called routine, the JSCI stack area is built on the current system stack. Otherwise, a stack switch takes place, as described in Section 3.10, and the save area is built on the stack associated with the process level of the called routine.

The S-bit, immediately following PPP, indicates a JSCI stack frame (S=0) or an SVC stack frame (S=1).

4.6.2 Linkage Table

The Linkage Table allows branching to external subroutines, which are dynamically linked to a program after it has been loaded into main memory. Each entry of the Linkage Table describes an external subroutine. The JSCI instruction refers only to the first 8 bytes of the a Linkage Table entry, whose format is shown in Figure 4-12. The total length of the entry minus 1 is indicated by the high-order byte of Control Register 6.

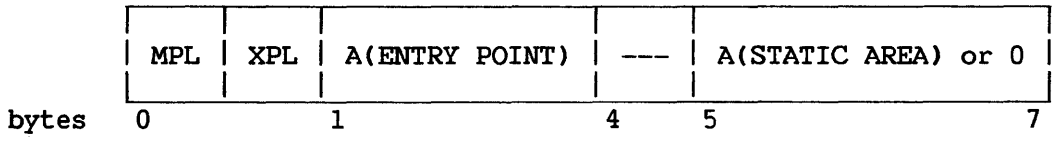


Figure 4-12. Format of Linkage Table Entry

MPL indicates the minimum process level required to call the given routine. XPL indicates the process level at which the called routine will execute. MPL and XPL are each 4-bit fields.

During the process of dynamic linking, the second operand of a JSCI instruction is resolved to the address of the Linkage Table entry that contains the entry point address of the called routine.

4.6.3 Control Registers 6 and 7

The location, length, and element size of a Linkage Table are indicated in Control Registers 6-7. The format of this information is shown in Figure 4-13.

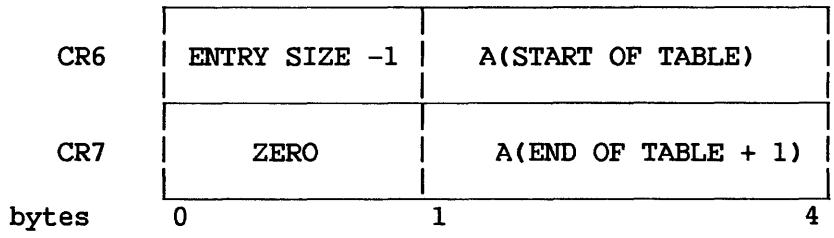


Figure 4-13. Control Registers 6-7

The Linkage Table entry size is indicated in bytes. The actual entry size (not the size-1) must be an integer power of 2.

4.6.4 Subroutine Branching

This section briefly describes the use of the Linkage Table by the JSCI instruction. For a more detailed description of JSCI execution, refer to Chapter 8, where the instruction is described.

The address specified by the second JSCI operand is compared with the range of addresses specified in CR6-7. If the branch address falls outside this range (and is a valid address), then the calling and called subroutines have been statically linked in the same load module, their process levels are the same, and stack switching is not required. The save area is built on the caller's stack, and control is passed to the called routine.

If the address specified in the JSCI instruction falls within the range of addresses in CR6-7, the current process level (CPL) is compared with the MPL and XPL values in the Linkage Table entry addressed by the JSCI instruction. The comparison of CPL to MPL ensures that the caller is privileged to execute the called routine. When CPL is equal to XPL, the save area is built on the caller's stack and control is passed to the called routine. Otherwise, a stack switch takes place, and the save area is built on the called routine's stack. The process level specified in the PCW (i.e., CPL) is raised to the XPL value, and control is passed to the called routine.

In branches to an external subroutine, General Register 14 is updated with a nonzero address of the subroutine's static area, taken from Bytes 5-7 of the Linkage Table entry.

CHAPTER 5
INTERRUPTIONS

5.1 INTRODUCTION

The interruption system permits the CP to change state as a result of conditions external to the system, in input/output (I/O) devices, or in the CP itself. Four classes of interruption conditions are possible:

- I/O
- Clock
- Program
- Machine check

Each class of interruption has two related PCWs called "old" and "new" in permanently assigned main memory locations. An interruption involves storing information, identifying the cause of the interruption, storing the current PCW in its old position, and making the PCW at the new position the current PCW. A new PCW is also assigned for use by the SVC instruction. The SVC, however, is more properly considered a special branching instruction than an interruption of the instruction stream. The SVC instruction itself, rather than an interrupt handler, copies the the new SVC PCW into the current PCW, as described in Chapter 8.

The old PCW holds necessary CP status information at the time of interruption. If, at the conclusion of the program invoked by the interruption, an instruction is executed making the old PCW the current PCW, the CP is restored to the state prior to the interruption, and the interrupted program continues.

5.2 POINT OF INTERRUPTION

An interruption is permitted between units of instructions, that is, after the performance of one instruction and before the start of a subsequent instruction. This is true for all instructions except interruptible instructions (MVCL, CLCL). Interruptible instructions can be interrupted during instruction performance. They resume from the point of instruction interruption after the interruption has been serviced.

5.2.1 Instruction Execution

An interruption occurs between instructions, except for interruptible instructions. The manner in which the preceding instruction is finished may be influenced by the cause of the interruption. The instruction is said to have been completed, terminated, aborted, suppressed, or resumed.

In the case of instruction completion (which is the usual case), results are stored and the condition code is set as for normal instruction operation, although the result may be influenced by the exception that has occurred.

In the case of instruction termination, all, part, or none of the result may be stored. Therefore, the result data is unpredictable. The setting of the condition code, if called for, may also be unpredictable. In general, the results should not be used for further computation. The PCW is not updated on termination.

When an instruction is aborted, all results including the condition code and the PCW are unpredictable. An instruction can be aborted only by a machine check interruption.

In the case of instruction suppression, results are not stored, the condition code is not changed, and the PCW is not updated. The condition code is indeterminate for suppressed floating-point instructions on machines with a hardware floating-point unit.

In the case of instruction resumption, the instruction resumes after a higher priority interruption has been serviced.

5.2.2 Location Determination

Nearly always, the instruction causing the interruption is given by the address in the PCW. In the rare case when an instruction is completed before the interruption occurs, the instruction address in the old PCW designates the next instruction to be executed.

5.3 MAIN MEMORY LOCATIONS

The four classes of interruptions are distinguished by the memory locations in which the old PCW is stored and from which the new PCW is fetched. The detailed causes are further identified by the interruption code stored in the first byte of the old PCW and in some cases by additional information placed in main memory during the interruption.

For I/O interruptions, additional information is provided by the contents of the I/O Status Word stored as part of the I/O interruption. (The I/O Status Word is discussed in Section 9.9.) For program interruptions caused by address translation exceptions, additional information may be provided in the form of a page index stored in the page fault reporting area and the address of the associated region node in the region node address area. For machine check interruptions, additional information may be stored in the machine check reporting area.

Table 5-1 lists the permanently allocated main memory locations.

Table 5-1. Permanent Storage Assignments

Address (hexadecimal)	Length (decimal)	Function
0	8	Input/Output Status Word (IOSW)
8	4	Reserved
C	20	Control mode communications area
20	8	Old PCW for machine check
28	8	New PCW for machine check
30	8	Old PCW for program check
38	8	New PCW for program check
40	8	Old PCW for clock interrupt
48	8	New PCW for clock interrupt
50	8	Old PCW for I/O interrupt
58	8	New PCW for I/O interrupt
60	8	New PCW for SVC
68	8	PCW save area (Control mode)
70	2	Reserved
72	2	Page fault reporting area
74	4	Region Table entry address
78	8	Machine check reporting area
80	2	I/O Device Address
82	14	For VS15 and VS65: CP/BP communications area For VS100: Reserved for system use
90	Variable	I/O status table

NOTE

The SVC Old PCW is placed on the system stack by means of the SVC instruction. It is reloaded (made current) from there by means of the SVCX instruction.

5.4 INPUT/OUTPUT INTERRUPTION

The I/O interruption provides a means by which the processor responds to signals from I/O devices.

A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests are preserved in the I/O device until accepted by the processor. While I/O interruptions are masked by setting the I/O interruption mask bit (Bit 37 of the Current PCW) to 0, more than one event which establishes a pending interruption may occur at a device. Each such event is recorded at the device, and when the I/O interruption mask bit is then set to 1, the I/O interruption for the device is taken. The stored I/O Status Word (IOSW) may reflect the occurrence of all such events by the ORing of status bits in the IOSW. Priority is established among devices so that only one interruption request is processed at a time.

An I/O interruption can occur only after the current unit of operation is finished and while the processor is interruptible. Interruptions not serviced remain pending.

The I/O interruption causes the Old PCW to be stored in the I/O Old PCW. The IOSW associated with the interruption will have been stored in the IOSW slot at the time of the interruption. Subsequently, a new PCW is loaded from the I/O New PCW.

5.5 CLOCK INTERRUPTION

The clock interruption provides a means by which the CP responds to timing conditions set within the system. Clock interruptions are maskable by zeroing the clock interruption mask bit (Bit 38 of the current PCW). Any clock interruption that becomes pending while the clock interruption mask bit is 0 remains pending. A pending clock interruption is taken immediately upon completion of any instruction that turns off the clock interruption mask bit in the PCW. The clock interruption causes the old PCW to be stored in the Clock Old PCW and a new PCW to be loaded from the Clock New PCW. The interruption code in the old PCW is set to all 0s on a clock interruption.

A clock interruption becomes pending whenever the time-of-day clock value is greater than or equal to the clock comparator value, both comparands being considered unsigned 64-bit binary quantities. The time-of-day clock is maintained in Control Registers 12-13; the clock comparator value is loaded in Control Registers 14-15.

Loading a comparator value that is already less than or equal to the time-of-day value causes an immediate interruption.

5.6 PROGRAM INTERRUPTION

Exceptions that result from improper use of instructions and data cause a program interruption. Only one program interruption occurs for a given instruction and is identified in the Old Program-Check PCW. The occurrence of a program interruption does not preclude the simultaneous occurrence of other causes of program interruption. The program interruption causes the current PCW to be stored at the Old Program-Check PCW location and a New Program-Check PCW to be fetched. The cause of the interruption is identified by the interruption code in PCW Bits 0-7. The operation is completed, suppressed, or terminated by a program interruption, but this is determined on an individual interruption basis.

A description of the individual program exceptions follows. Some of the exceptions listed may also occur in operations resulting from I/O instructions. In such cases, the exception is indicated in the IOSW stored with the I/O interruption (as explained in Section 9.9.2).

5.6.1 Program Interruption Codes in the PCW

Program interruption codes are defined as follows:

Program Interruptions	Hex Code
Programming Errors and Miscellaneous Exceptions	
Operation	01
Privileged operation	02
Execute	03
Protection	04
Addressing	05
Specification	06
Data	07
Fixed-Point overflow	08
Fixed-Point divide	09
Decimal overflow	0A
Decimal divide	0B
Supervisor call range	0C
Load-or-trap	0D
Debug Facility	
Debug trap taken	10
Debug trap specification	16
Address Translation Exceptions	
Page fault	20
Region table entry fault	21
Page table address recursion	22
Region table entry range	25
Translation structure	26
Paging File I/O Error (software-defined error code)	28
Unresolved External Reference (software-defined error code)	29

Stack Facility	
Stack overflow	30
Stack header block	36
Floating-Point Exceptions	
Floating-Point overflow	40
Floating-Point underflow	41
Significance	42
Floating-Point divide	43

5.6.2 Access Exceptions

The protection, addressing, debug trap, and address exceptions are collectively referred to as access exceptions. An access exception may be indicated when a reference to a partially inaccessible operand is recognized even if the correct result could be arrived at without the use of the inaccessible part of the operand. The access exception is indicated as part of the execution of the instruction making the reference.

Whenever an access to an operand location can cause an access exception to be recognized, the word "access" is included in the list of program exceptions in the description of the instruction. This entry also indicates which operand can cause the exception to be recognized and whether the exception is recognized on a fetch or store access to that operand location. Also, each instruction can cause an access exception to be recognized due to instruction fetch.

An access exception is indicated only if the instruction with which the exception is associated is executed. Thus, the exception is not recognized when

- The CP has not attempted a fetch from the inaccessible location or otherwise detected the access exception before a branch instruction.
- The interruption changes the instruction sequence such that the inaccessible data is not required.

5.6.3 Operation Exception

When an operation code is not assigned, an operation exception is recognized. The first eight bits of an instruction are considered to form the operation code.

5.6.4 Privileged-Operation Exception

A privileged instruction or operation is defined to be one that generates an exception if the user mode bit (Bit 34) of the PCW is on. Some VS privileged instructions are CIO, HIO, LCTL, LPCW, RRCB, STNSM, STOSM, SIO, STDD, and SVCX. When a privileged instruction is encountered while this bit is on in the PCW and the current process level is not 7, a privileged-operation exception is recognized, and the instruction is suppressed.

5.6.5 Execute Exception

The execute exception is recognized when the subject instruction of EXECUTE is another EXECUTE. The instruction is suppressed.

5.6.6 Protection Exception

A protection exception is recognized when all of the following conditions obtain:

- The PCW memory protection bit (PCW Bit 34) is on.
- The current process level is not 7.
- Fetch or store access is disallowed by the protection bits of a valid T-RAM entry or by the read-minimum or write-minimum fields of a region table entry.

5.6.7 Addressing Exception

When an address specifies any part of a datum, an instruction, or a control word to be outside the user's address space, an addressing exception is recognized. On a branch instruction or any instruction that introduces a new PCW, the address to which control is to be passed is not checked for validity; thus, the addressing exception will occur on the instruction that was branched to and not on the branch instruction itself. An addressing exception always causes instruction termination.

5.6.8 Specification Exception

A specification exception is recognized when any of the following conditions exist:

- An operand address does not designate a location on a doubleword, fullword, or halfword boundary, depending on the instruction type.
- The first operand field is shorter than or equal to the second operand field in decimal division.
- An invalid head/tail queue word has been specified in enqueue/dequeue operations.
- Other special cases exist.

5.6.9 Data Exception

A data exception is recognized when either of the following conditions occurs:

- The digit codes of operands in decimal arithmetic or editing operations or in CONVERT TO BINARY are incorrect
- Fields in decimal arithmetic overlap incorrectly.

5.6.10 Fixed-Point Overflow Exception

When a high-order carry occurs or high-order significant bits are lost in fixed-point add, subtract, arithmetic shift, or sign-control operations, a fixed-point overflow is recognized. When an overflow occurs and the corresponding mask bit is set to 1, the exception is recognized.

5.6.11 Fixed-Point Divide Exception

A fixed-point divide exception is recognized when either of the following situations occur:

- The quotient exceeds the register size in fixed-point division, including division by 0
- The result of CONVERT TO BINARY exceeds 31 bits.

5.6.12 Decimal Overflow Exception

When the receiving field is too small in a decimal arithmetic operation, a decimal overflow is recognized. When an overflow occurs and the corresponding mask bit is set to 1, the exception is recognized.

5.6.13 Decimal Divide Exception

A decimal divide exception is recognized when the quotient in decimal division exceeds the specified data size.

5.6.14 Supervisor Call Range Exception

Issuance of a SUPERVISOR CALL (SVC) instruction with a value in the I operand field greater than the value in the first byte of the Supervisor Call New PCW results in a supervisor call range exception, and the instruction is suppressed.

5.6.15 Load or Trap Exception

A load or trap exception is recognized when the LOAD OR TRAP (LOT) instruction has loaded a fullword field from memory into a general register and the high-order bit of the loaded word is equal to 1.

5.6.16 Debug Facility Exceptions

Debug Trap Taken Exception

A Debug trap taken exception (X'10') occurs when, immediately after processing of the Debug table, the trap taken flag (TTF) of any active entry is found to be set to 1. A description of the Debug table and traps is provided in Chapter 7.

Debug Trap Specification

A Debug trap specification is recognized under any of the following conditions: The Debug table spans a page, the Debug table is not word aligned, an entry in the table is not in the order prescribed for its trap type, or TTF flags set during previous processing of the table have not been cleared prior to current processing of the table.

5.6.17 Address Translation Exceptions

Five address translation exceptions are recognized; they are described in the following paragraphs. In the case of page fault and page table address fault exceptions, the page index of the faulted address and the address of the associated region table entry are written, respectively, in the page fault reporting area (Location X'72') and the region table entry address area (Location X'74').

- Page Fault Exception -- A page fault exception (Code X'20') occurs when the page table entry corresponding to the virtual address is faulted; i.e., when its high order bit (fault bit) is 1. This is an ordinary page fault, and causes the virtual page to be read in from disk storage.

- Page Table Address Fault Exception -- A page table address fault exception (Code X'21') occurs when the page table address reported in the appropriate region table entry is virtual and is faulted.

- Page Table Address Recursion Exception -- A page table address recursion exception (Code X'22') occurs when the page table address reported in the appropriate region table entry is virtual and the region table entry accessed to translate the page table address reports another virtual page table address. In this case, the second virtual page table address is not translated and an exception is noted immediately.

- Region Table Entry Range Exception -- A region table entry range exception (Code X'25') occurs when a virtual address falls within the range of a region table, but not within the LOHI range of any entry in that region table.

- Translation Structure Exception -- A translation structure exception (Code X'26') occurs when the address of a region table reported in a segment control register is zero, a region table is not word-aligned, or the page table address reported in a region table entry is zero.

5.6.18 Stack Facility Exceptions

Stack Overflow Exception

The stack overflow exception (Code X'30') occurs under either of the following conditions:

- The address value in the stack top word is less than the address value in the stack limit word before the instruction is executed.

- The address value in the stack top word would be less than the address value in the stack limit word after the instruction was executed.

The instruction is suppressed on all stack overflow program interrupts. This implies that the values in the stack vector are unchanged.

Stack Header Block Exception

A stack header block exception (Code X'36') occurs under either of the following conditions:

- A stack header block address (reported in the stack header pointer table) is zero.

- The address of a new stack header block, found by indexing into the stack header pointer table, is the same as the address of the current stack header block, reported in Control Register 8.

5.6.19 Floating-Point Exceptions

Four kinds of floating-point exceptions are recognized; they are described in the following paragraphs.

- Floating-Point Overflow -- When the final exponent of a floating-point number becomes greater than 127 as a result of an ADD, SUBTRACT, MULTIPLY, or DIVIDE operation, the instruction is completed and a floating-point overflow exception is recognized. The fraction is correct and normalized if normalization was specified by the instruction, the sign is correct, and the characteristic is smaller by 128 than the correct characteristic.

- Floating-Point Underflow -- When the final exponent of a floating-point number becomes less than zero as a result of an ADD, SUBTRACT, MULTIPLY, DIVIDE, or HALVE operation, and the exponent underflow program mask bit is 1, the instruction is completed and a floating-point underflow exception is recognized. The fraction is correct and normalized if normalization was specified by the instruction, the sign is correct, and the characteristic is larger by 128 than the correct characteristic.

- Floating-Point Significance -- When the intermediate sum of a floating-point ADD or SUBTRACT operation is zero, and the significance program mask bit is 1, a significance exception is recognized. No normalization occurs; the intermediate sum characteristic remains unchanged. When the intermediate sum is zero and the significance program mask bit is 0, the significance exception does not occur; rather, the characteristic is made zero, yielding a true zero result.

- Floating-Point Divide -- A floating-point divide exception is recognized when floating-point division by a divisor with a fraction of zero is attempted. The instruction is suppressed and the dividend remains unchanged.

5.7 MACHINE CHECK INTERRUPTION

The machine check interruption provides a means for reporting machine malfunctions so that processing can halt and corrective action be taken.

An enabled machine check interruption causes the old PCW to be stored in the Machine Check Old PCW and a new PCW to be fetched from the Machine Check New PCW. The cause of the malfunction is identified by an interruption code stored in the Old PCW. On the VS300, the interruption code and an explanatory message are displayed at the System Control Unit (SCU) running in Control mode. On other VS systems, the interruption code is displayed in decimal notation at Workstation 0.

5.7.1 Memory error

A multibit memory cell error causes a machine check when the error cannot be corrected by the associated Error Correction Check (ECC) code. When a read or write operation by the CP generates a memory error, a cell isolation routine (CIR) attempts to locate the failing cell. Successful CIR determines the physical address of the failing cell and the erroneous data. On the VS300, this information is stored in the Memory Control Unit (MCU). It can be displayed at the SCU by means of the Machine Status utility. On other VS systems, the erroneous data is placed in main memory Location X'78' and the physical address of the failing cell (i.e., word) is placed in Location X'7C'; otherwise, Location X'7C' contains a zero address.

During an I/O operation, memory errors are detected by the I/O processor and reported in the I/O Status Word, as described in Section 9.9.2.

5.7.2 Interrupt Codes

Listed below are the machine check interrupt codes placed in the Old Machine Check PCW. On the VS300, the codes are displayed in hexadecimal at the SCU running in Control mode; on other VS systems, they are in decimal at workstation 0.

Interrupt Codes--VS15, VS65, VS100

Decimal Code	Hex Code	Definition
001	01	Main Memory parity error on read by CP. Old PCW contains the address of the instruction following the instruction which received the error and then aborted. CIR used. X'78' contains erroneous data; X'7C' contains word address of error. Address is zero if CIR failed. VS15 and VS65 This error may be detected on a 1-byte read-modified-write operation.
002	02	IOP transmit error. After requesting and then receiving permission to present an interruption, the IOP made another request or presented the interruption after expiration of timeout. Byte 0 of the machine check reporting area contains device address, or X'FF' in case of timeout.
003	03	Main memory error on a 1-byte read-modified-write. Old PCW contains the address of the instruction following the instruction which caused the error and then completed. CIR used. X'78' contains erroneous data; X'7C' contains word address of error. Address is zero if CIR failed.
017	11	Bus Transaction log overflow. The old PCW contains the address of the instruction following the instruction which caused the error.
018	12	IOP receive error. IOP has rejected CP or BA communication. Old Machine Check PCW points to the instruction whose execution followed (but did not necessarily cause) the error.
019	13	Both errors X'11' and X'12' have occurred.
020	14	IOP receive error. IOP has rejected CP communication. Old Machine Check PCW points to the instruction whose execution followed (but did not necessarily cause) the error.
021	15	Both errors X'11' and X'14' have occurred.
022	16	Both errors X'12' and X'14' have occurred.
023	17	Errors X'11', X'12', and X'14' have occurred.

Interrupt Codes--VS300

Hex Code	Definition
01	Main Memory parity error on read by CP. Old PCW contains the address of the instruction following the instruction which received the error and then aborted. CIR used. The SCU can read the address of the failing cell along with erroneous data. This information resides in the MCU; it is not stored by the CPU in the machine check reporting area.
0F	Default trap taken; probable hardware error within the CP card set.
10	Address Generation Unit (AGU) error received. Invalid state in instruction queue. Probable hardware error within the AGU.
20	I/O interrupt received with no active IOSW in the IOC status table. This signals a failure within the CP/IOC protocol.
21	Power fail interrupt received. The system is running on battery backup. The clock (Control Registers 12-13) is stored at Locations X'78' and X'7C' immediately upon receipt of the power fail indication (after the current instruction is completed).
22	Spare control exception trap taken; probable hardware failure within the CP card set.
41	Translation buffer parity error; probable hardware error within the CP card set.
42	Illegal state -- external cache probe. Probable Address Translation Unit (ATU) hardware error.
44	Illegal state -- internal cache probe. Probable ATU hardware error.
50	System bus parity error. Probable hardware error in any system bus element (i.e., SCU, System Bus Interface (SBI), ATU, MCU, or backplane).

5.8 PRIORITY OF INTERRUPTIONS

5.8.1 Overview

Some interrupts are recognized, or detected, during instruction execution; others are recognized between execution of instructions.

The first class of interrupts are recognized and handled as they occur; there is no established priority for handling them. The second class of interrupts, however, are recognized and handled in an order that is independent of their occurrence. During the execution of an instruction, several of these interrupts can occur; at the end of instruction execution, they are all outstanding. At this time, the CP checks for these interruptions in a certain order, which is independent of their occurrence and is the reverse of the order in which they are actually handled, once detected.

When multiple interrupts are outstanding between instruction execution, the following actions occur: After one of these interruptions is detected, a swap of PCWs takes place, after which the current PCW points to the interrupt handler for the detected interruption. Before execution of the instruction addressed by the current PCW, interruptions (of the second class) are again checked for, except those of the type(s) already detected since execution of the last instruction. Upon detection of another interruption, another swap of PCWs occurs, after which the current PCW addresses the handler for the interrupt last detected. This process is repeated until no interruptions are outstanding. Then, the handler of the interruption last detected is executed; the old PCW associated with the interruption serves as a back chain to the handler of the interruption detected next-to-last. Thus, the order in which interrupt handlers execute is the reverse of the order in which interrupts are detected.

By setting bits of the PCW status field, interrupt handlers can disable interrupts to prevent the contents of old PCW areas from being overwritten. While thus masked out, interrupts except machine checks are ignored; when disabled machine checks occur, the system enters Control mode.

5.8.2 Priority of Detection

Listed below, in order of their detection and in reverse order of their handling, are interrupts detected between instruction execution.

- Program check (Interrupt Code X'05'--addressing exception)
- Machine check
  X'03'
  X'1x'
- Clock interrupt
- I/O interrupt

CHAPTER 6
CONTROL MODE

6.1 INTRODUCTION

6.1.1 Control Mode Facilities

Control mode is a CP state in which normal program execution is halted and certain other facilities are made available. These facilities are implemented by two groups of commands.

1. Load commands -- Commands for loading into main memory the operating system or a standalone program. (These commands are not supported by the VS15 and VS65 systems.)

2. Debug commands -- Commands for displaying and/or modifying main memory, general registers, control registers, and the PCW. Also included in this group are commands for single-step program execution and virtual address translation. All VS processors support debug commands.

On all systems except the VS300, Control mode commands are entered at Workstation 0; on the VS300, Control mode commands are entered at the Support Control Unit (SCU) running in Console mode. For the the sake of conciseness, this chapter generally refers to both Workstation 0 and the SCU in Console mode as the system console. It should be noted, however, that the two terminals are fundamentally different in that Workstation 0 communicates with the CP and memory controller through intermediate processors (shown in Figures 1-1 through 1-3), whereas the SCU communicates directly with system elements through the support packet bus.

6.1.2 Control Mode Communications Area

The Control mode communications area (X'C'-X'1F') is a main memory area used for communications between the CP and console. Through the communications area, the CP instructs the device to enter Control mode. The CP receives, via the communications area, Control mode commands entered at the console, and then places requested data and status information in the area.

When Control mode is entered from program execution, the current PCW is saved in the last eight bytes (X'18' to X'1F') of the Control mode communications area. In leaving Control mode, the CP picks up the PCW from this location.

6.2 METHODS OF ENTRY

Control mode can be entered at the time of initial program load (IPL) or during program execution.

After entry from IPL, load or debug commands are available. Load commands, when successfully executed, cause the loading and execution of IPL text, which can be a bootstrap loader of the operating system, a standalone diagnostic, or any other VS code.

When Control mode is entered during program execution, debug commands are available.

6.2.1 Entry from IPL (VS100)

The following sequence of steps taken during system initialization cause the system to enter Control mode:

1. Powering up the system.

2. Pressing the BT (bootstrap) button on the control panel to load the CP microcode.

3. Pressing the LOAD or INITIALIZE button on the control panel. Pressing the LOAD button initializes T-RAM to make low memory addressable, and clears the other translation structures and main and cache memory. Pressing the INITIALIZE button additionally initializes the system clock (Control Registers 12-13) and comparator (Control Registers 14-15).

At Step 3, the system enters Control mode.

When the system is re-initialized, after it has been powered up and the CP microcode has been loaded, the following sequence of steps cause entry into Control mode:

1. Pressing the CM (Control mode) button on the control panel.

2. Pressing the LOAD or INITIALIZE button.

At Step 2, the system enters Control mode.

6.2.2 Entry from IPL (VS300)

System initialization -- Pressing the ON button powers up the system, loads the CP microcode, initializes system elements, and causes the system to enter Control mode. Initiation and completion of these processes is indicated on the SCU Start-up screen. After all processes are completed, the SCU accepts Control mode commands.

System re-initialization -- After the system has been powered up and the CP microcode has been loaded, pressing the RESET button on the control panel clears memory and causes the CP to enter Control mode. Control mode commands can be entered after selecting Console mode from the SCU Main menu and then Control mode from the Console menu.

6.2.3 Entry During Program Execution

All VS systems enter Control mode during program execution after any of the following events:

- The Control mode bit (Bit 33) of the current PCW has been set by an instruction such as LPCW or STOSM.
- The Control mode button on the control panel has been pushed; or, in the case of the VS300, Control mode has been selected from the console menu.
- A single step trap, set from Control mode, has been taken.
- A machine check has occurred when the M-bit of the PCW (Bit 39) has been masked, disabling machine check interrupts.

6.3 LOAD COMMANDS

Load commands select a fixed or removable disk platter from which IPL text is to be loaded; entering the commands also initiates the loading process. The commands are available only after Control mode has been entered at IPL time, as described in Sections 6.2.1 and 6.2.2.

Load commands identify the IPL device by specifying numbers for the following:

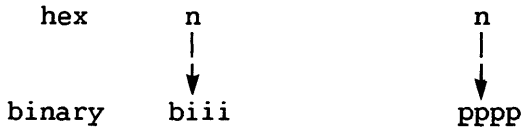
VS100	VS300
Bus Adapter (BA)	System Bus Identifier (SBI)
I/O Processor (IOP)	I/O Controller (IOC)
Port	Port

The BA/SBI, IOP/IOC, and port numbers of a device can be found in the file @CONFIG@ in the library @SYSTEM@ on the system volume.

On the VS15 and VS65 systems load commands are not supported. Instead, the IPL device is manually selected by a hardware switch, and loading is initiated from a menu on Workstation 0.

6.3.1 VS100 Load Commands

R nn Initiates loading of IPL text from the removable (R) disk platter identified by nn. These hexadecimal values indicate the bus adapter, IOP, and port of the IPL device, as follows:



where b = 0 for BA 1
1 for BA 2

iii = IOP 0-7

pppp = device 0-15

For example, a removable platter on port 4 of IOP 3 on BA 2 would be indicated in binary as:

(biii)	(pppp)
1011	0100

and in hexadecimal as:

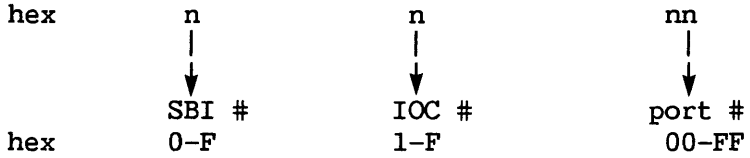
B	4

The command for loading from this device is: R B4

F nn Initiates loading of IPL text from the fixed (F) disk platter identified by nn. These hexadecimal values indicate the BA, IOP, and port of the fixed platter as for the R command.

6.3.2 VS300 Load Commands

R nnnn Initiates loading of IPL text from the removable (R) disk platter identified by nnnn. These hexadecimal values indicate the system bus interface, IOC, and port of the IPL device, as follows:



Although the architecture supports hexadecimal values from 0 through F for the SBI number, only the value of 0 is currently allowed. This value, specifying the first SBI, is assumed if three or fewer digits are entered.

For the IOC number, specified by the second digit, a value of 1 is assumed if two or fewer digits are entered.

A port number of one or two digits must be entered; there is no default port number.

F nnnn — Initiates loading of IPL text from the fixed (F) disk platter identified by nnnn. These hexadecimal values indicate the SBI, IOC, and port of the fixed platter just as they do when appearing as the argument of the R command.

6.3.3 Execution of Load Commands -- VS100

After a load command has been entered, the CP places an IOCW in an I/O Command Table (IOCT) entry starting at Location 200 (hexadecimal). Each entry is 16 bytes in length; so, the starting location of the IOCT, in hexadecimal, is

200 - (p * F)

where p is the port number specified with the Load command. The CP stores the command table address (CTA) in that entry of the IOP Status Table (IOPST) which corresponds to the IOP specified in the Load command. The IOPST starts at Location 90 (hexadecimal).

The IOCW specifies a read of 2048 bytes from sector 0 of the IPL device into main memory, starting at Location 800 (hexadecimal).

In successful execution of a Load command, the CP issues an SIO command to the IPL device, awaits an I/O interrupt from the device, inspects the I/O Status Word (IOSW) returned by the device after the read operation completes, and passes control to the IPL text.

The IOSW is stored by the device in the I/O status table, which starts at main memory Location 90 (hexadecimal). I/O status tables, the IOSW, IOCW, IOCT, and CTA are described in Chapter 9.

VS100 Error Messages

Listed and explained below are the error messages displayed at the console when load group commands do not execute successfully.

Error Message	Meaning
INV DEV	Either the IOP rejected the SIO sent by the CP, or the IOP specified by the Load command does not exist.
INT REQ	The SIO was accepted, but the device returned an Intervention Required IOSW.
I/O ERROR	The SIO was accepted, but the device returned an Error Completion IOSW. The IOSW can be inspected at Location X'0' for a determination of the I/O error.

6.3.4 Execution of Load Commands -- VS300

After a load command has been entered, the SCU places an IOCW in that entry of an I/O Command Table (IOCT) which corresponds to the port number specified in the command. The IOCT starts at Location 280 (hexadecimal). The CP stores this command table address (CTA) in that entry of the IOC Status Table (IOCST) which corresponds to the IOC specified in the load command. The IOCST starts at Location 90 (hexadecimal).

The IOCW specifies a read of 2048 bytes from sector 0 of the IPL device to main memory, starting at Location 800 (hexadecimal).

After issuing a START I/O (SIO) command to the specified IOC, the SCU waits for the IOC to interrupt. When an interrupt is detected, the SCU checks each IOCST entry for a Status Qualifier Byte (SQB) whose IOSW Active Flag bit is set to 1. The entry with this bit set corresponds to the interrupting IOC.

The Physical Device Address (PDA) stored in this entry is compared with the PDA specified by the Load command. If the PDAs do not match, the interrupt was not from the expected IOC (i.e., the IOC specified in the command). Unexpected interrupts are cleared and ignored. The SQB of the entry is examined for an error condition. Any error is reported and the SIO is reissued.

When the returned IOSW is found to originate from the IOC specified in the Load command, the IOSW is examined for indications of an unsolicited interrupt and error completion. Unsolicited interrupts are cleared and ignored. Error completions are reported, and the IPL process is restarted.

When a solicited, normal completion IOSW is received from the IOC specified in the Load command, the SCU stores the command code of the IOCW in Location 20 (hexadecimal) and stores the specified PDA in Location 80 (hexadecimal). The IPL text later uses the command code and PDA to continue loading from the disk.

The SCU, using the Control mode communications area, sets the PCW instruction address to the start of the IPL text (IAD 800, hexadecimal). The SCU then exits from Control mode, and the IPL text begins execution.

The IOCW, IOCT, IOCST, PDA, and SQB are described in Chapter 9.

VS300 Error Messages

Listed and explained below are the error messages displayed at the console when load commands do not execute successfully.

Error Messsage	Meaning
IPL Failed. No SQB received from the IOC	The IOC specified by the Load command did not respond to the command. The specified IOC or device may not exist.
IPL Failed. Invalid SQB	The IOC stored an invalid SQB and may be faulty.
IPL Failed. No interrupt received from IOC	An interrupt was not received from the IOC after it stored an SQB. This indicates a system bus failure.
IPL failed. Drive not ready or I/O error	IOC returned an Error Completion IOSW. The IPL drive is not powered on, the disk may be damaged, or the drive may be faulty.

6.4 VS15, VS65, AND VS100 DEBUG COMMANDS

G n — Displays general registers n and n+1, with n ranging from 0 to E. (A value of n=F results in display of general register F followed by general register 0.)

C n — Displays control registers n and n+1, with n ranging from 0 to F.

P nnnnnn — Displays the data in physical address nnnnnnn.

V nnnnnn — Translates the virtual address nnnnnn by means of the LPA instruction. Displays the condition code and the contents of R1 resulting from an LPA instruction. The condition code comprises the first two digits of the displayed numbers. If the translation is successful (condition code = 00), the condition code is followed by a three-byte physical address and eight bytes of memory starting at that physical address. In the event of unsuccessful translation, a nonzero condition code is followed by a 3-byte address consisting of zeros.

W — Displays the PCW.

M — Allows modification of the eight bytes displayed as a result of the G or C commands. When entered, this command does not appear on the screen; the cursor moves to the first modifiable character. A character is modified by being overstruck. Pressing the space bar passes over a character without modifying it. Not all eight characters need be modified; but any characters between the first and last ones modified must be overstruck or spaced over.

TAB (key) Causes execution of a single program step and displays the updated PCW. All I/O operations will proceed normally.

X Causes exit from Control mode; instruction execution proceeds under control of the current PCW.

NOTE

In Control mode no device other than Workstation 0 will be serviced by the Control mode I/O processor (i.e., keystrokes from other workstations are ignored).

For the V command, a failure to display data may indicate that a page break (2048-byte boundary) has been found; otherwise, nondisplay (or a partial display) indicates that a main memory parity error was detected by the IOP at the particular memory location.

For the single step command, re-entry into Control mode with the PCW not updated indicates that the single stepped instruction caused a T-RAM fault. The single step command can be successfully entered again at this point.

Change bits in the Reference and Change Table (RCT) are not set for any pages modified by a P or V command followed by an M command.

6.5 VS300 DEBUG COMMANDS

The E, P, S, and V commands display on the SCU screen the contents of the PCW, eight bytes of memory starting at the current instruction address, general registers, control registers, floating-point registers, and 128 bytes of main memory. The argument of the E, P, or V commands specifies the starting address of the 128-byte display. If a V or P command has not set the address mode, the starting address is physical.

E expr — Examines memory starting at address specified by "expr", where "expr" may be one of the following:

```
addr
sym
sym+addr
sym-addr
```

where

addr — A 6-digit hexadecimal number

sym — The symbol $, signifying the last byte of currently displayed memory; or the symbol PC, signifying the current instruction address

The most recent execution of the P or V command causes the argument "expr" to be treated, respectively, as a physical or virtual address. In the absence of a P or V command, the address mode defaults to physical.

L — After an M command, positions the cursor to the first floating point register.

M — Allows the modification of data displayed on the screen by the invocation of other debug commands.

M addr — Allows the modification of data at the address specified by "addr". The mode of "addr" -- physical or virtual -- is set respectively by the last invocation of a command that specifies P or V (i.e., a P, V, MVaddr, or MPaddr command). In the absence of a command specifying P or V, the mode defaults to P.

MPaddr — Allows the modification of data at the physical address specified by "addr".

MVaddr — Allows the modification of data at the virtual address specified by "addr".

P addr — Displays memory starting at the physical address specified by "addr", where "addr" is a 6-digit hexadecimal number.

S [or] TAB — Causes execution of a single program step and displays the updated PCW. All I/O operations will proceed normally.

V addr	Displays memory starting at the virtual address specified by "addr", where "addr" is a 6-digit hexadecimal number.
T	Causes the SCU, but not the CP, to exit from Control mode.
X	Causes the SCU and the CP to exit from Control mode; instruction execution proceeds under control of current PCW.

NOTE

Change bits in the Reference and Change Table (RCT) are not set for pages modified by any form of the M command.

6.5.1 PF Keys for E Command

Two frequently-used E commands can be entered with a single keystroke as follows:

PF4	Equivalent to "E $-80"; displays previous 128 bytes.
PF5	Equivalent to "E $+80"; displays next 128 bytes.

6.5.2 Cursor Control Keys for M Command

After the M command has been entered, the following keys can be used to position the cursor to or within modifiable fields. Modifiable fields are highlighted; they comprise all fields except the eight bytes starting at the current instruction address.

Key	Function
Cursor Right, Left Up, Down Arrow.	Moves cursor anywhere on the screen.
TAB	Moves cursor to start of next modifiable field.
BACKTAB	Moves cursor to start of current field if not already there; else, to start of preceding field.
HOME	Moves cursor to first modifiable field, which is PCW.
SPACE	Advances cursor to next modifiable position without altering data, or to a new field if necessary.

Key	Function
BACKSPACE	Backs up cursor to preceding modifiable position without altering data, or to a new field if necessary.
P	Positions cursor to PCW data.
G	Positions cursor to start of general register data (i.e., start of general register 0).
S	Positions cursor to start of control register data (i.e., start of control register 0).
M	Positions cursor to the first modifiable byte displayed on the Control mode screen.

6.6 ENTERING AND CANCELLING COMMANDS

The RETURN key must be pressed to execute a Control mode command.

On VS100 systems, a command that has not been entered may be cancelled and cleared by the BACKSPACE key. On the VS300 system, cancellation of a command is effected by the CANCEL key.

6.7 EDITING COMMAND LINES

On the VS300 system only, command lines may be edited. Editing is accomplished by the following keys:

Key	Function
BACKSPACE or Cursor Left Arrow	Acts as a rubout key; moves the cursor one position left and erases any character in the position on which the cursor rested before the move.
Cursor Right Arrow	Moves the cursor one position right, restoring any erased character in the position on which the cursor rested before the move.
BACKTAB [or] HOME	Positions the cursor at the start of the command line.

6.8 CONTROL MODE DUMPS

The procedure for taking system dumps in Control mode is described in the VS System Operator's Guide.

CHAPTER 7
DEBUG FACILITY

7.1 DEBUG FACILITY OVERVIEW

The Debug facility comprises features of VS machine architecture intended to support debugging utilities. The principal structure of the facility is the Debug table, in which a debugging utility can define the conditions under which traps are to be taken (i.e., process execution is to be interrupted). The table supports six trap types, each with its own entry format. The DBG bit of the PCW (Bit 40), qualified by control information in Control Registers 5 and 10, activates processing of the Debug table.

Processing of the Debug table takes place after checking of the DBG bit and control registers, and before execution of the next program instruction. During processing, every entry in the table is inspected. When a trap condition defined by an active entry (Entry Active bit = 1) is found to be satisfied, the trap taken flag (TTF) bit of the entry is set to 1. Processing continues uninterrupted until all entries of the table are examined. Then, if the TTF bit of any entry has been set to 1, a trap taken exception (Code X'10') is recognized.

The flow of Debug table processing is described more fully in Section 7.4.

7.2 TRAP TYPES

Table entries for all traps, except the memory modification trap, include a 4-bit trap identifier. A memory modification trap entry is indicated by the setting of a single bit. Supported trap types and their identifier are as follows:

Trap Identifier	Trap Type
-	Memory Modification
1	PCW Trap
2	Instruction Step
3	Opcode Trap
4	PCW Range Trap
5	General Register Modification

The function of each trap type is indicated in Section 7.7.2.

7.3 CONTROL REGISTER 3

The operating system places in Control Register 3 the address of the Debug table and the number of table entries, as illustrated in Figure 7-1.

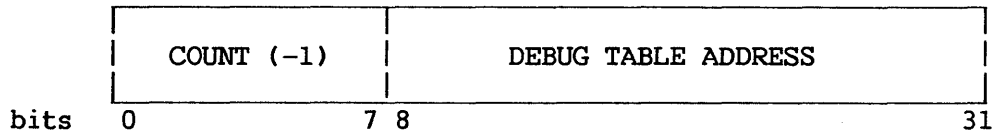


Figure 7-1. Format of Control Register 3

The value of the register's high-order byte indicates the number of table entries minus one. The address provided by the remaining bytes is virtual.

7.4 CONTROL REGISTERS 4 AND 5

The CP places in Control Registers 4 and 5, respectively, the prior and current instruction address at the time that the trap exception is taken. In addition, the CP uses the high order byte of Control Register 5 as a synchronization flag that is cleared at the outset of table processing and set at the completion of processing. The format of this register pair is shown in Figure 7-2.

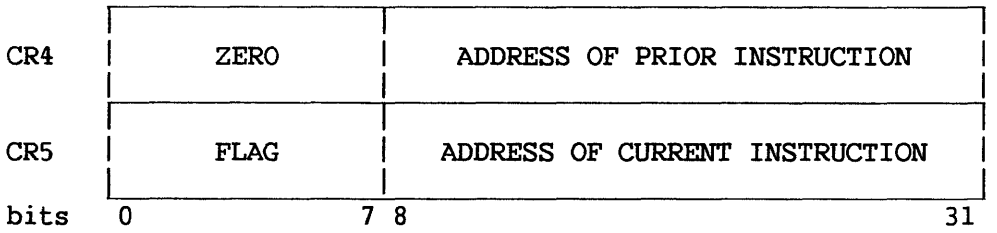


Figure 7-2. Format of Control Registers 4 and 5.

Before the processing of table entries, the instruction address in Control Register 5 is moved to Control Register 4. Then the current instruction address (i.e., the address of the instruction next to be executed) is copied from the PCW to Control Register 5. Each table entry is inspected; if the trap condition(s) defined by the entry are met, the entry's TTF bit is set to 1. Control Register 4 points to the instruction that caused the trap. After all table entries have been inspected, the flag byte is set to X'80'.

At the time that the DBG bit is inspected, it is possible that the current instruction (i.e., the instruction next to be executed) had already started executing, was interrupted, and is about to be restarted. In this case, the Debug table has already been processed for the current instruction. The values of the flag byte and current instruction address in Control Register 5 are used to determine whether, in fact, the table has already been processed for the current instruction. This determination is necessary to maintain the accuracy of counter values in table entries for trap types 1-5. The flag byte, unlike TTF bits, must not be cleared by the debugging utility.

7.5 CONTROL REGISTER 10

The low-order byte of Control Register 10 holds the values of two process levels. The two 3-bit values, each left-justified in a 4-bit field, signify the inclusive range wherein Debug traps are active. When the current process level is outside this range, the DBG bit of the PCW is ignored.

7.6 TABLE FORMAT

The Debug table must be word-aligned. It cannot span pages; thus, because each table entry is 12 bytes in length, the table cannot comprise more than 170 entries.

The table can include any mix of trap type entries in any order, except that any entry(ies) for the memory modification trap must precede entries for other traps, and may not be intermixed with entries for other types.

7.7 TABLE ENTRY FORMAT

7.7.1 Format of Byte 0

There is a unique table entry format for each trap type. However, as indicated by Table 7-1, the format of the first byte is common to trap types 1-5, and the format of the first four bits is common to all trap types.

Table 7-1. Format of First Byte of Debug Table Entries

Bit Number	Function
Trap Types 0-5	
0	Entry Active 0 = Entry inactive 1 = Entry active
1	Trap Taken Flag (TTF) 0 = Trap not taken 1 = Trap taken
2	Reserved
3	Memory Modification Select 0 = Trap type is 1-5 1 = Trap type is 0 (memory modification)
Trap Type 0	
4	Comparison 0 = Trap when operand and comparand are unequal 1 = Trap when operand and comparand are equal
5-7	Operand length A value of 0-7; length of operand in bytes minus 1
Trap Types 1-5	
4-7	Trap Type 1 = PCW trap 2 = Instruction step 3 = Opcode trap 4 = PCW Range trap 5 = General Register Modification trap

7.7.2 Format of Bytes 1-11

Main Memory Modification Trap

Figure 7-3 shows the format of entries for the main memory modification trap (trap type 0).

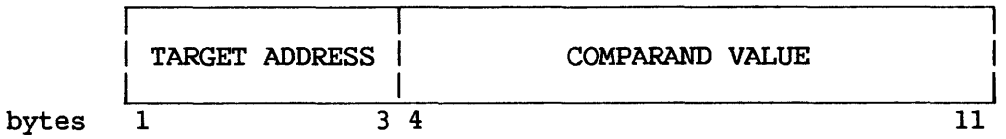


Figure 7-3. Entry for Main Memory Modification Trap

The target address in Bytes 1-3 indicates the virtual address of the operand in memory. The length of this operand is indicated by the Bits 5-7 of Byte 0 (described in Table 7-1). The comparand value in Bytes 4-11 is considered to be left-adjusted. A logical comparison between the operand and comparand value is performed. Then the comparison bit in Byte 0 is inspected to determine whether the trap condition is equality or inequality of operand and comparand. The TTF bit in Byte 0 is set when the specified condition is met.

PCW Trap

Figure 7-4 shows the format of entries of the PCW trap (trap type 1)

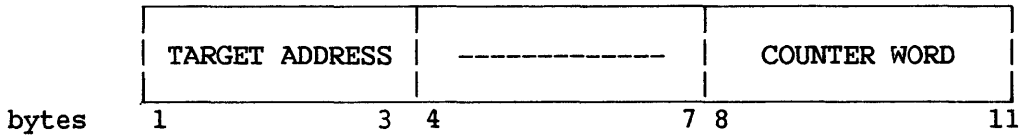


Figure 7-4. Entry for PCW Trap

The virtual target address in Bytes 1-3 is compared with the instruction address in the current PCW. In the case of equivalence, the value of the counter word is decremented by 1. If the value of the decremented counter word is 0, the TTF bit in Byte 0 is set to 1.

Instruction Step Trap

Figure 7-5 shows the format of entries for the Instruction Step trap (trap type 2).

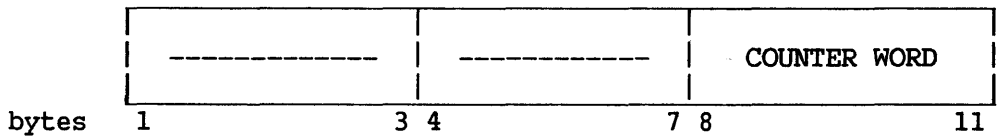


Figure 7-5. Entry for Instruction Step Trap

The value of counter word is decremented by 1. If the decremented value is 0, the TTF bit in byte 0 is set to 1.

Opcode Trap

Figure 7-6 shows the format of entries for Opcode traps (trap type 3).

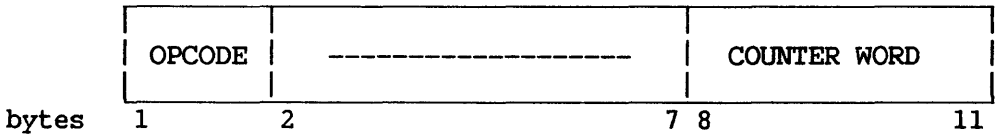


Figure 7-6. Entry for Opcode Trap

The value of the opcode field in Byte 1 is compared with the opcode byte of the current instruction. In the case of equivalence, the value of the counter word is decremented by 1. If the value of the decremented counter word is 0, the TTF bit in Byte 0 is set to 1.

PCW Range Trap

Figure 7-7 shows the format of entries for the PCW Range trap (trap type 4).

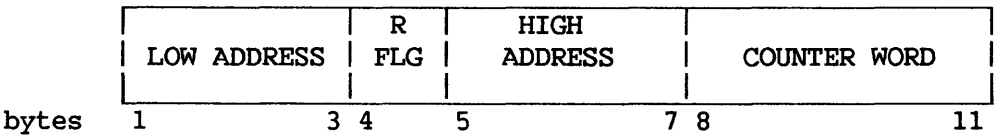


Figure 7-7. Entry for PCW Range trap

This entry specifies as a trap condition either the inclusion or exclusion of the current PCW instruction address by a specified range of addresses.

The low address field (Bytes 1-3) and high address field (Bytes 5-7) indicate the address range. A PCW instruction address greater than or equal to the low address, and less than or equal to the high address is considered to be in-range.

The first bit of Byte 4 serves as the range flag. A flag value of 0 means "trap if in-range condition is met"; a value of 1 means "trap if in-range condition is not met". If the trap condition is met, the value of the counter word is decremented by 1. Decrementing to 0 causes the TTF flag bit in Byte 0 to be set to 1.

General Register Modification Trap

Figure 7-8 shows the format of entries for the General Register Modification trap (trap type 5).

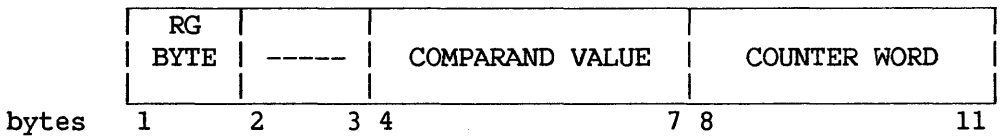


Figure 7-8. Entry for General Register Modification Trap

The Register (RG) byte has the following format:

Bits 0-3 Mask:
X'8' Trap on equivalence
X'4' Trap if comparand is low
X'2' Trap if comparand is high

Bits 4-7 Number of General register

The comparand (Bytes 4-7) is compared with the contents of the general register indicated by the RG byte (Bits 4-7). If the outcome of the comparison meets the condition indicated by the RG mask (Bits 0-3), the counter word value is decremented by 1. Decrementing to 0 causes the TTF flag bit in Byte 0 to be set to 1.

7.8 COUNTER WORD

The counter word does not wrap around at 0. If the value of the counter word is 0 before decrementing takes place, the value remains zero and the TTF bit of the entry is set to 1.

The Debug table is processed and the counter decremented before instruction execution. Thus, to indicate a trap condition of n iterations, the counter value must be n+1. For example, the stepping of a single instruction is accomplished by using a counter value of 2.

CHAPTER 8
INSTRUCTIONS

8.1 GENERAL INSTRUCTION SET

The following instructions represent the basic instruction set for the VS. In addition to these universal machine instructions, the assembler supports some extended mnemonic codes, such as JSI, which are discussed in the chapter on machine instructions in the VS Assembly Language Reference. A list of operation codes and formats for the basic instruction set is provided in Appendix A of this manual.

The superscript "p" (e.g., $(CIO)^p$) in the first line of an instruction description means that the instruction is privileged.

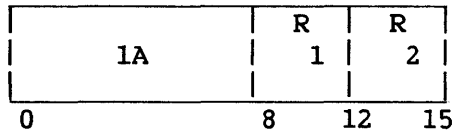
Instructions are ordered alphabetically by name in this chapter, and alphabetically by mnemonic in the index.

Instructions having extended operand codes (i.e., 9Bxx) are described in Section 8.2.

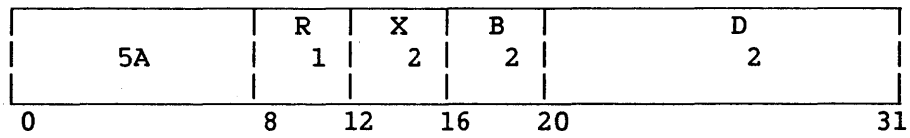
The order in which program exceptions are listed, within each instruction description, is not necessarily the order in which they occur. When multiple exceptions are pending, the order in which they are recognized depends on the type of VS system being used.

ADD (AR, A)

AR R1,R2 (RR)



A R1,D2(X2,B2) (RX)



The second operand is added to the first operand, and the sum is placed in the first operand location.

Addition is performed by adding all 32 bits of both operands. If the carry from the sign-bit position and the carry from the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit in the PCW is 1.

Operand 2 of the A instruction must be fullword aligned.

Resulting Condition Code

0 Sum is 0
1 Sum is less than 0
2 Sum is greater than 0
3 Overflow

Program Exceptions

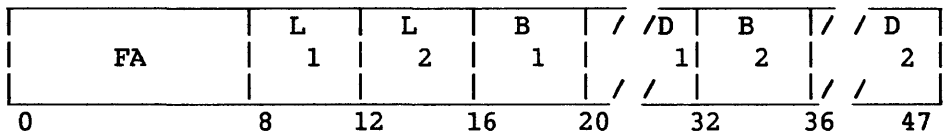
Access (fetch, operand 2 of A only)
Fixed-point overflow
Specification (A only)

Programming Note

In 2's-complement notation a zero result is always positive.

ADD DECIMAL (AP)

AP D1(L1,B1),D2(L2,B2) (SS)



The second operand is added to the first operand, and the sum is placed in the first operand location.

L1 and L2 are the field lengths in bytes, minus 1.

Addition is algebraic, taking into account sign and all digits of both operands. All digits are checked for validity. If necessary, 0s are supplied for either operand on the most significant end. When the first operand field is too short to contain all significant digits of the sum, an overflow condition is recognized.

Overflow has two possible causes. The first is the loss of a carry from the most significant digit position of the result field. The second cause is an oversized result, which occurs when the second operand field is larger than the first operand field and significant result digits are lost. The field sizes alone are not an indication of overflow. An overflow causes a program interruption when the decimal overflow mask bit is 1.

The first and second operand fields may overlap when their least significant bytes coincide; therefore, it is possible to add a number to itself.

The sign of the result is determined by the rules of algebra. When the operation is completed without an overflow, a zero sum result has a positive sign, but when high-order digits are lost because of an overflow, a zero result may be either positive or negative, as determined by what the sign of the correct result would have been. This instruction will set the condition code even if the decimal overflow exception is taken.

Resulting Condition Code

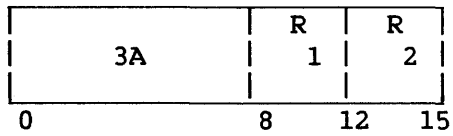
0 Sum is 0
1 Sum is less than 0
2 Sum is greater than 0
3 Overflow

Program Exceptions

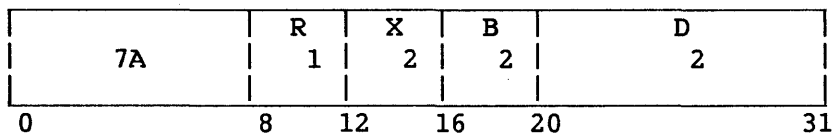
Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal overflow

ADD DECIMAL (FLOATING-POINT) (AQR, AQ)

AQR R1,R2 (RR)



AQ R1,D2(X2,B2) (RX)



The second operand is added to the first operand, and the normalized sum is placed in the first operand location. Fullword alignment is required.

Addition of two decimal floating-point numbers consists of a characteristic comparison and a fraction addition. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted; its characteristic is increased by one for each decimal digit of shift until the two characteristics agree. The fractions are then added algebraically to form an intermediate sum. If an overflow carry occurs, the intermediate sum is right-shifted one digit and the characteristic is increased by one. If the increase causes a characteristic overflow, a program interruption occurs. The fraction and the sign are correct, but the characteristic is 128 smaller than the correct characteristic.

The intermediate sum consists of 15 decimal digits and a possible carry. The low-order digit is a guard digit obtained from the fraction that is shifted right. The guard digit is 0 if no shift occurs.

After the addition, the intermediate sum is normalized as necessary by shifting left the fraction; vacated low-order digit positions are filled with 0s; the characteristic is reduced by the amount of shift.

If normalization causes the characteristic to underflow and if the corresponding mask bit is 1, a program interruption occurs. The fraction is correct, but the characteristic is 128 larger than the correct one. If the corresponding mask bit is 0, the result is made a true zero.

When the intermediate sum is zero and the significance mask bit is 1, a significance exception exists and a program interruption takes place. No normalization occurs; the intermediate sum characteristic remains unchanged. When the intermediate sum is zero and the significance mask bit is 0, the program interruption for significance exception does not occur; rather, the result is forced to be true zero. Exponent underflow cannot occur for a zero fraction.

The sign of the sum is derived by the rules of algebra. A zero sum fraction is regarded as positive.

Resulting Condition Code

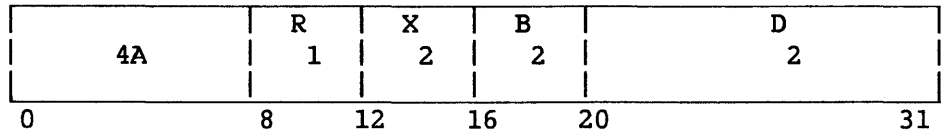
0	Result fraction is 0
1	Result fraction is less than 0
2	Result fraction is greater than 0
3	-

Program Exceptions

Specification
Data
Significance
Exponent overflow
Exponent underflow
Access (AQ only)

ADD HALFWORD (AH)

AH R1,D2(X2,B2) (RX)



The second operand is added to the first operand, and the sum is placed in the first operand location. The second operand is two bytes in length, must be halfword aligned, and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the addition by propagating the sign-bit value through the 16 high-order bit positions. The contents of the second operand in main memory remain unchanged. Addition is performed by adding all 32 bits of both operands. If the carry from the sign-bit position and the carry from the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The overflow causes a program interruption when the fixed-point overflow mask bit in the PCW is 1.

Resulting Condition Code

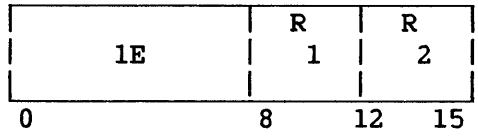
0 Sum is 0
1 Sum is less than 0
2 Sum is greater than 0
3 Overflow

Program Exceptions

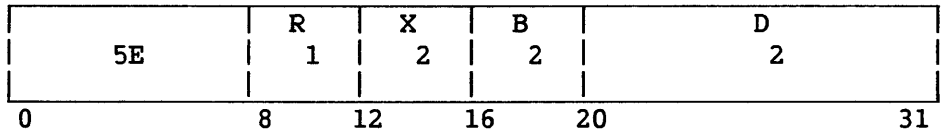
Access (fetch, operand 2)
Fixed-point overflow
Specification

ADD LOGICAL (ALR, AL)

ALR R1,R2 (RR)



AL R1,D2(X2,B2) (RX)



The second operand is added to the first operand, and the sum is placed in the first operand location. The occurrence of a carry from the sign position is recorded in the condition code.

The second operand of the AL instruction must be fullword aligned.

Logical addition is performed by adding all 32 bits of both operands. If a carry from the leftmost position occurs, the leftmost bit of the condition code is made 1. In the absence of a carry, the leftmost bit is made 0. When the sum is 0, the rightmost bit of the condition code is made 0. A nonzero sum is indicated by a 1 in the rightmost bit.

Resulting Condition Code

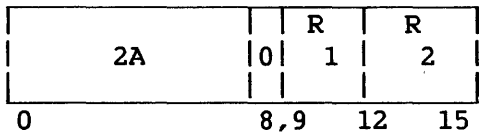
0 Sum is 0 (no carry)
1 Sum is not 0 (no carry)
2 Sum is 0 (carry)
3 Sum is not 0 (carry)

Program Exceptions

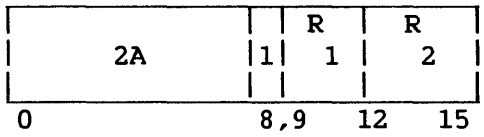
Access (fetch, operand 2 of AL only)
Specification (AL only)

ADD NORMALIZED (FLOATING-POINT) (ADR, AER, AD, AE)

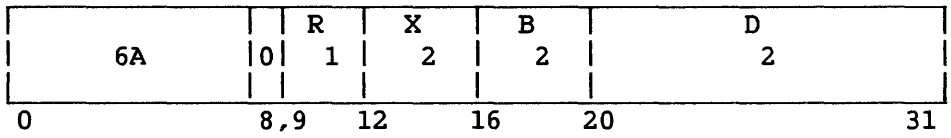
ADR R1,R2 (RR, Long)



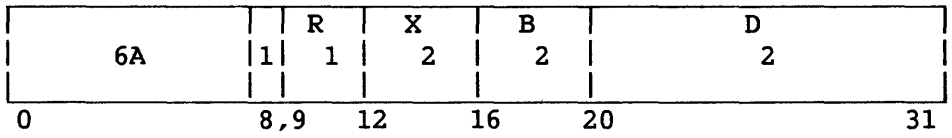
AER R1,R2 (RR, Short)



AD R1,D2(X2,B2) (RX, Long)



AE R1,D2,(X2,B2) (RX, Short)



The second operand is added to the first operand, and the normalized sum is placed in the first operand location.

Operand 2 of the AD instruction must be fullword aligned.

Addition of two floating-point numbers consists of comparing characteristics and adding fractions. The characteristics of the two operands are compared, and the fraction with the smaller characteristic is right-shifted; its characteristic is increased by 1 for each hexadecimal digit of shift until the two characteristics agree. The fractions are then added algebraically to form an intermediate sum. If an overflow carry occurs, the intermediate sum is right-shifted one digit and the characteristic is increased by 1. If this increase causes a characteristic overflow, an exponent-overflow exception is signaled and a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is smaller by 128 than the correct characteristic.

The intermediate sum consists of 15 hexadecimal digits (for AER and AE, 7 hexadecimal digits) and a possible carry. The low-order digit is a guard digit obtained from the fraction that is shifted right. Only one guard digit position participates in the fraction addition. The guard digit is 0 if no shift occurs.

After the addition, the intermediate sum is left-shifted as necessary to form a normalized fraction, vacated low-order digit positions are filled with 0s, and the characteristic is reduced by the amount of shift.

If normalization causes the characteristic to underflow and if the corresponding mask bit is 1, a program interruption occurs. The fraction is correct and normalized, the sign is correct, and the characteristic is larger by 128 than the correct one. If the corresponding mask bit is 0, the result is made a true 0. If no left shift takes place, the intermediate sum is truncated to the proper fraction length.

When the intermediate sum is 0 and the significance mask bit is 1, a significance exception exists, and a program interruption takes place. In this case, no normalization occurs; the intermediate sum characteristic remains unchanged. When the intermediate sum is 0 and the significance mask bit is 0, the program interruption for the significance exception does not occur; rather, the characteristic is made 0, yielding a true zero result. Exponent underflow does not occur for a fraction of 0.

The sign of the sum is derived according to the rules of algebra; a result of 0 is regarded as positive.

Resulting Condition Code

0 Result fraction is 0
1 Result is less than 0
2 Result is greater than 0
3 --

Program Exceptions

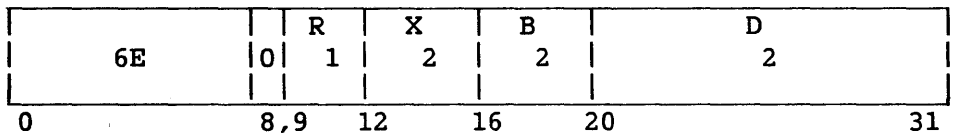
Specification
Significance
Exponent overflow
Exponent underflow
Access

Programming Note

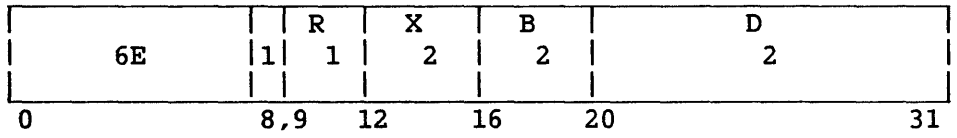
Interchanging the two operands in a floating-point addition does not affect the value of the sum.

ADD UNNORMALIZED (FLOATING-POINT) (AW, AU)

AW R1,D2(X2,B2) (RX, Long)



AU R1,D2(X2,B2) (RX, Short)



The second operand is added to the first operand, and the unnormalized sum is placed in the first operand location. Operand 2 requires fullword alignment.

After the addition the intermediate sum is truncated to the proper fraction length.

When the resulting fraction is 0 and the significance mask bit in the PCW is 1, a significance exception exists and a program interruption takes place. When the resulting fraction is 0 and the significance mask bit is 0, the program interruption for the significance exception does not occur; rather, the characteristic is made 0, yielding a true zero result. (See ADD NORMALIZED.)

Leading 0s in the result are not eliminated by normalization, and an exponent underflow cannot occur.

The sign of the sum is derived by the rules of algebra. The sign of a sum with a result fraction of 0 is always positive.

Resulting Condition Code

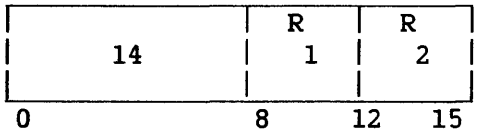
0 Result fraction is 0
1 Result is less than 0
2 Result is greater than 0
3 --

Program Exceptions

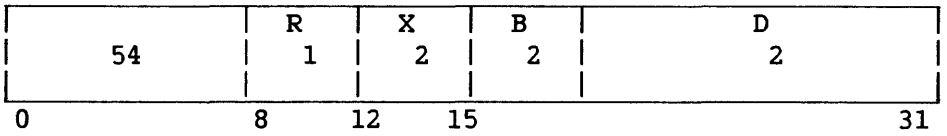
Specification
Significance
Exponent overflow
Access

AND (NR, N, NI, NC)

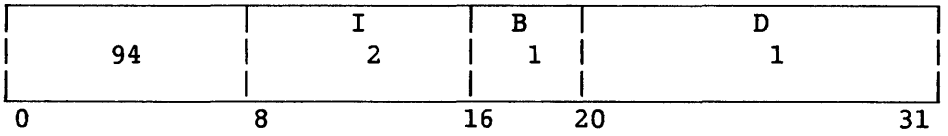
NR R1,R2 (RR)



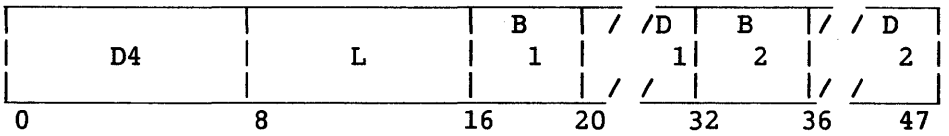
N R1,D2(X2,B2) (RX)



NI D1(B1),I2 (SI)



NC D1(L,B1),D2(B2) (SS)



The logical product (AND) of the bits of the first and second operand is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit positions in both operands contain a 1; otherwise, the result bit is set to 0. All operands and results are valid.

Operand 2 of the N instruction must be fullword aligned. For the NC instruction, L is the length of each operand minus 1.

Resulting Condition Code

0 Result is 0
1 Result not 0
2 --
3 --

Program Exceptions

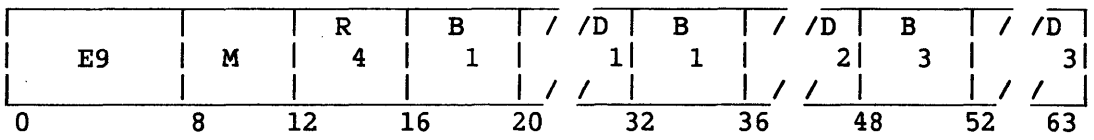
Access (fetch, operand 2, N and NC; fetch and store, operand 1, NI, NC)

Specification (N only)

Programming Note

The AND instruction may be used to set a bit to 0. For this purpose, the second operand should have 0s in all positions corresponding to the first-operand bits to be set to 0.

ACHECK R4,D1(B1),D2(B2),D3(B3) (Special)



Access to the memory area defined by the first and second operands is checked; the result is placed in the condition code.

The first operand addresses the first byte of the area to which access is to be verified. The second operand specifies the length of the area in bytes. The third operand specifies the process level at which access to the area is to be verified. The fourth operand, if nonzero, specifies a register which, if the defined area spans two regions, receives the starting address of the second region.

The 4-bit mask field determines the type of access to be verified. Valid values for this field are X'8' for read access, X'4' for write access, and X'2' for execute access.

Read access is allowed (mask = X'8' and returned condition code = 0) as follows:

- When the specified process level is greater than 0, read access is allowed if 1) the defined area falls in LOHI range of the first region node in the region table addressed by the first segment Control Register (SCR 0), or 2) the minimum read level of the region is less than or equal to the specified process level.

- When the specified process level is 0, read access is allowed if 1) the minimum read level of the region is 0, and 2) no part of the defined area lies in the stack area for process level zero between the current stack pointer and the beginning of the heap area.

Write access is allowed (mask = X'4' and returned condition code = 0) as follows:

- When the specified process level is greater than 0, write access is allowed if the minimum write level for the region that includes the defined area is less than or equal to the specified process level.

- When the specified process level is 0, write access is allowed if 1) the minimum write level of the region is 0, and 2) no part of the defined area lies in the stack area for process level zero between the current stack pointer and the beginning of the heap area.

Execute access is allowed (mask = X'2' and returned condition code = 0) for process levels 0-7 if minimum read level of the region that includes the defined area = 0 and the minimum write level = 7.

When the mask value = X'8' or X'4' and the specified process level = 0, the value of the stack pointer (General Register 15) is obtained from the stack header block for process level zero. A specification error occurs if this stack header block spans memory pages.

A specification error occurs if the specified process level is greater than 7 or greater than the current process level.

Resulting Condition Code

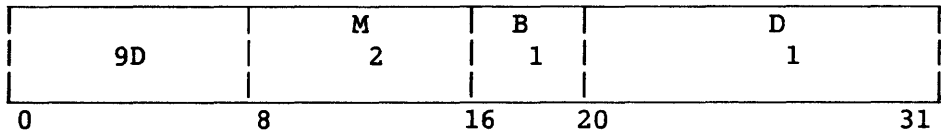
0 Access type specified is allowed for this area and process level
1 Area crosses a region boundary (first part allowed)
2 Region not found (or specification error)
3 Region found but access denied

Program Exceptions

Privileged operation
Specification

BIT RESET (BRESET)

BRESET D1(B1),M2 (SI)



The bit at bit displacement M2 from the high-order bit (Bit 0) of the first operand is set to 0. Bit numbering begins with the high-order bit of each byte and proceeds through ascending byte locations. The condition code reflects the value of the specified bit before modification.

Resulting Condition Code

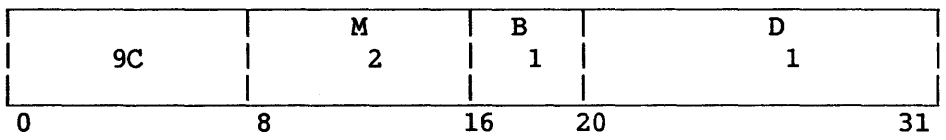
0 Bit was 0 before operation
1 Bit was 1 before operation
2 --
3 --

Program Exceptions

Access (store, operand 1)

BIT SET (BSET)

```
BSET   D1(B1),M2                (SI)
```



The bit at bit displacement M2 from the high-order bit (Bit 0) of the first operand is set to 1. Bit numbering begins with the high-order bit of each byte and proceeds through ascending byte locations. The condition code reflects the value of the specified bit before modification.

Resulting Condition Code

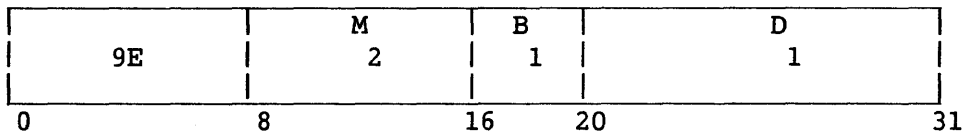
0 Bit was 0 before operation
1 Bit was 1 before operation
2 --
3 --

Program Exceptions

Access (store, operand 1)

BIT TEST (BTEST)

```
BTEST   D1(B1),M2              (SI)
```



The bit at bit displacement M2 from the high-order bit (Bit 0) of the first operand is tested, and the result is reflected in the condition code. Bit numbering begins with the high-order bit of each byte and proceeds through ascending byte locations.

Resulting Condition Code

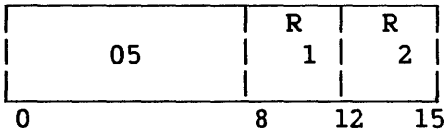
0	Bit is 0
1	Bit is 1
-	--
-	--

Program Exceptions

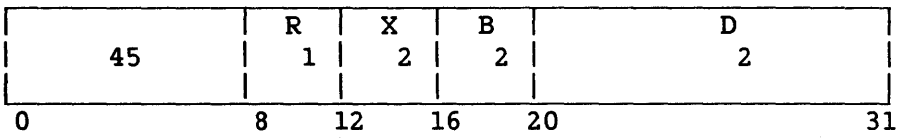
Access (fetch, operand 1)

BRANCH AND LINK (BALR, BAL)

BALR R1,R2 (RR)

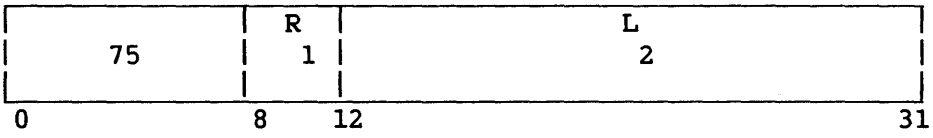


BAL R1,D2(X2,B2) (RX)



BRANCH AND LINK (RELATIVE) (RBAL)

RBAL R1,L2 (RL)



The program mask byte of the PCW and the updated instruction address are stored as link information in the general register specified by R1. Subsequently, the instruction address is replaced by the branch address. For BALR, the branch address is the contents of R2; for BAL, it is X2+B2+D2. For RBAL, the branch address is the sum of the current instruction address and the L2 field.

The branch address is determined before the link information is stored. The link information contains the updated instruction address.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

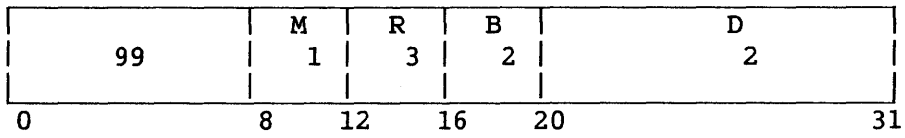
None

Programming Note

The link information is stored without branching in the RR format when the R2 field contains zero.

BRANCH AND LINK ON CONDITION INDIRECT (BALCI)

BALCI M1,R3,D2(B2) (RS)



The updated instruction address is replaced by the branch address if the state of the condition code is as specified by M1; otherwise, normal instruction sequencing proceeds with the updated instruction address. If the branch is taken, the program mask byte of the PCW and the updated instruction address are stored as link information in the general register specified by R3.

The branch address is determined before the link information is stored. The three low-order bytes of the word at the location designated by the second operand address are used as the branch address.

The M1 field is used as a 4-bit mask. The four bits of the mask correspond, left to right, with the four condition codes are as follows:

Instruction Bit	Mask Position Value	Condition Code
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the condition code has a corresponding mask bit of 1.

Operand 2 requires fullword alignment.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

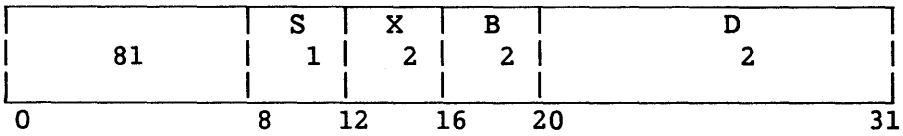
Access (fetch, operand 2 if the branch is taken)
Specification

Programming Note

This instruction combines a conditional branch and link with an indirectly specified branch address.

BRANCH AND LINK STACK (BALS)

BALS S1,D2(X2,B2) (RX)



The relevant stack vector is determined from the S1 field of the instruction. A branch address is calculated from the second operand field according to the rules for base-displacement or relative address formation. The stack pointer is decremented by 4, and the same information BAL would put in a register, including the updated instruction address, is placed in the four byte locations starting with the location addressed by the updated stack pointer. A branch is made to the previously calculated branch address.

Resulting Condition Code

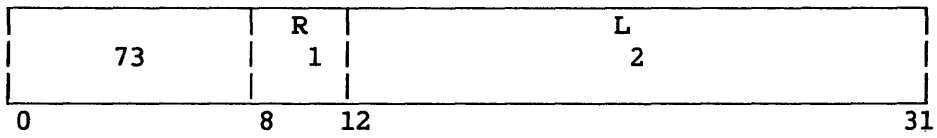
The condition code remains unchanged.

Program Exceptions

Stack overflow
Access (store, bytes pushed onto stack)
Specification

BRANCH AND LINK STACK (RELATIVE) (RBALS)

RBALS R1,L2 (RL)



The relevant stack vector is determined from the R1 field of the instruction. A branch address is calculated as the sum of the current instruction address and the L2 field. The stack pointer is decremented by 4, and the same information BAL would put in a register, including the updated instruction address, is placed in the four byte locations starting with the location addressed by the updated stack pointer. A branch is made to the previously calculated branch address.

Resulting Condition Code

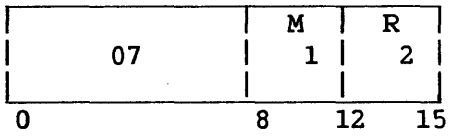
The condition code remains unchanged.

Program Exceptions

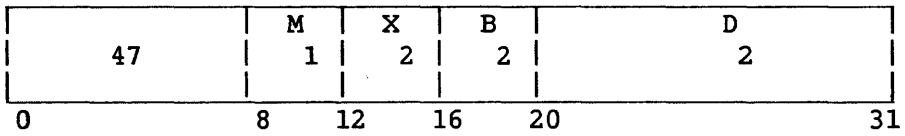
Stack overflow
Access (store, bytes pushed onto stack)

BRANCH ON CONDITION (BCR, BC)

BCR M1,R2 (RR)

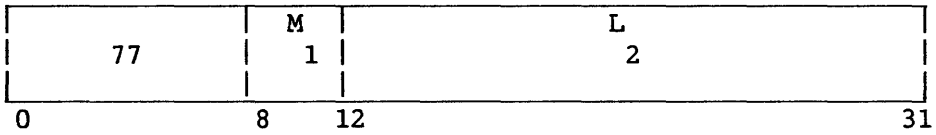


BC M1,D2(X2,B2) (RX)



BRANCH ON CONDITION (RELATIVE) (RBC)

RBC M1,L2 (RL)



The updated instruction address is replaced by the branch address if the state of the condition code is as specified by M1; otherwise, normal instruction sequencing proceeds with the updated instruction address. For BCR, the branch address is contained in R2; for BC, it is X2+B2+D2. For RBC, it is the sum of the current instruction address and the L2 field.

The M1 field is used as a 4-bit mask. The four bits of the mask correspond, left to right, with the four condition codes as follows:

Instruction Bit	Mask Position Value	Condition Code
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the condition code has a corresponding mask bit of 1.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

None

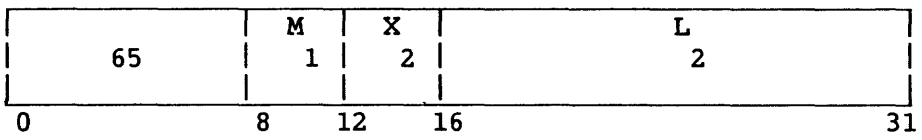
Programming Notes

When a branch is to be made on more than one condition code, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made on condition codes 0 or 1.

When all four mask bits are 1s, that is, when the mask position value is 15, the branch is unconditional. When all four mask bits are 0s or when the R2 field in the RR format contains 0, the branch instruction is equivalent to a no-operation. For a no-operation BCR the branch address (R2) is ignored.

BRANCH ON CONDITION INDEXED (RELATIVE) (RBCX)

RBCX M1,L2(X2) (RRL)



The updated instruction address is replaced by the branch address if the state of the condition code is as specified by M1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The branch address is formed by adding the halfword L2 field, the contents of the general register designated by the X2 field, and the current instruction address.

The M1 field is used as a 4-bit mask as in the BRANCH ON CONDITION (BC) instruction.

When the instruction is executed, the current instruction address used in the effective-address calculation is the address of the EXECUTE instruction.

Resulting Condition Code

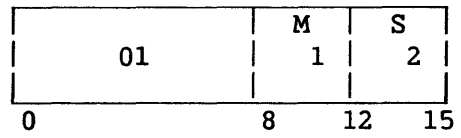
The condition code remains unchanged.

Program Exceptions

None

BRANCH ON CONDITION STACK (BCS)

BCS M1,S2 (RS)



The updated instruction address is replaced by the branch address if the state of the condition code is as specified by M1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The M1 field is used as a 4-bit mask. The four bits of the mask correspond, left to right, with the four condition codes as follows:

Instruction Bit	Mask Position Value	Condition Code
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the condition code has a corresponding mask bit of 1.

The relevant stack vector is determined from the S2 field of the instruction. The 24-bit address in the low-order three bytes of the word-aligned 4-byte memory area addressed by the contents of the stack pointer is placed in the current instruction address field in the PCW (i.e., a branch is made to that location). The stack pointer is then incremented by 4.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

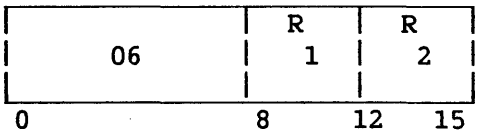
Access (fetch, bytes popped from stack)
Specification

Programming Note

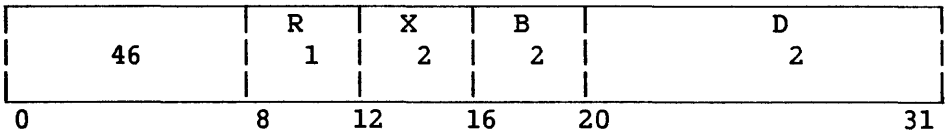
This instruction is a BCR that uses the stack.

BRANCH ON COUNT (BCTR, BCT)

BCTR R1,R2 (RR)



BCT R1,D2(X2,B2) (RX)



The contents of the general register specified by R1 are algebraically reduced by 1. When the result is 0, normal instruction sequencing proceeds with the updated instruction address. When the result is not 0, the instruction address is replaced by the branch address. The branch address for BCTR is R2; for BCT it is X2 + B2 + D2.

The branch address is determined prior to the counting operation. Counting does not change the condition code. The subtraction proceeds as in fixed-point arithmetic, and all 32 bits of the general register participate in the operation.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

None

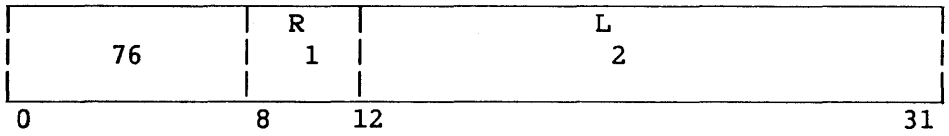
Programming Notes

An initial count of 1 results in 0, and no branching takes place. An initial count of 0 results in all 1s and causes branching to be executed.

Counting is performed without branching when the R2 field in the RR format contains 0.

BRANCH ON COUNT (RELATIVE) (RBCT)

RBCT R1,R2 (RL)



The sign of the L2 field is extended 12 bits to the left, to form a 32-bit signed 2's-complement displacement. The displacement is added to the current instruction address to form the branch address.

Instruction execution is then identical to the corresponding RX instruction.

When the instruction is executed, the current instruction address used in the effective-address calculation is the address of the EXECUTE instruction.

Resulting Condition Code

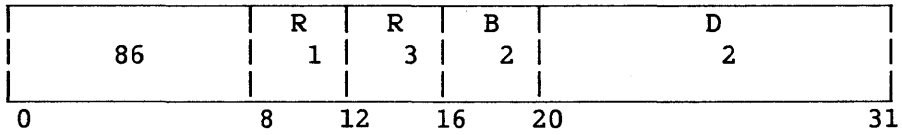
The condition code remains unchanged.

Program Exceptions

None

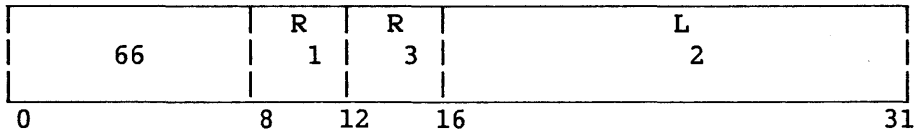
BRANCH ON INDEX HIGH (BXH)

BXH R1,R3,D2(B2) (RS)



BRANCH ON INDEX HIGH (RELATIVE) (RBXH)

RBXH R1,R3,L2 (RRL)



An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first operand location, regardless of whether the branch is taken. When the sum is high, the instruction address is replaced by the branch address. When the sum is low or equal, instruction sequencing proceeds with the updated instruction address. For BXH, the branch address is B2+D2. For RBXH, it is the sum of the current instruction address (bits 8-31 of the PCW) and the L2 field.

The first operand and the increment are in the registers specified by R1 and R3. The comparand register address is odd and is either greater by 1 than R3 or equal to R3. The branch address is determined prior to the addition and comparison.

Overflow caused by the addition is ignored and does not affect the comparison. Otherwise, the addition and comparison proceed as in fixed-point arithmetic. All 32 bits of the general registers participate in the operations, and negative quantities are expressed in 2's-complement notation. When the first operand and comparand locations coincide, the original register contents are used as the comparand.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

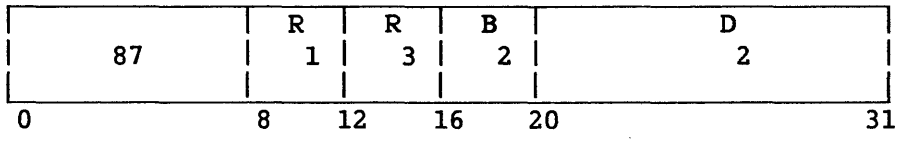
None

Programming Note

The name "branch on index high" indicates that one of the major purposes of this instruction is the incrementing and testing of an index value. The increment is algebraic and may be of any magnitude.

BRANCH ON INDEX LOW OR EQUAL (BXLE)

BXLE R1,R3,D2(B2) (RS)



An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first operand location, regardless of whether the branch is taken. When the sum is low or equal, the instruction address is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address. The branch address is B2+D2.

The first operand and the increment are in the registers specified by R1 and R3. The comparand register address is odd and is either greater by 1 than R3 or equal to R3. The branch address is determined prior to the addition and comparison.

This instruction is similar to BRANCH ON INDEX HIGH, except that the branch is taken when the sum is low or equal compared to the comparand.

Resulting Condition Code

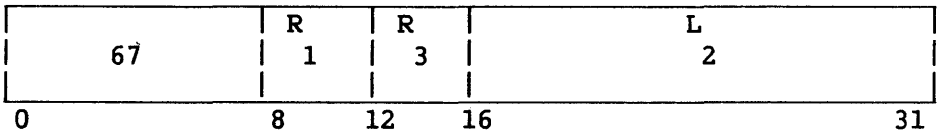
The condition code remains unchanged.

Program Exceptions

None

BRANCH ON INDEX LOW OR EQUAL (RELATIVE) (RBXLE)

RBXLE R1,R3,L2 (RRL)



The branch address is formed by adding the signed halfword L2 field and the current instruction address. Instruction execution is then identical to the corresponding RS instruction.

When the instruction is executed, the current instruction address used in the effective-address calculation is the address of the EXECUTE instruction.

Resulting Condition Code

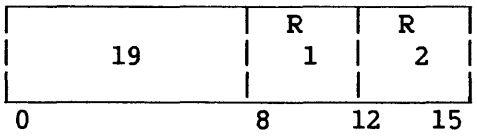
The condition code remains unchanged.

Program Exceptions

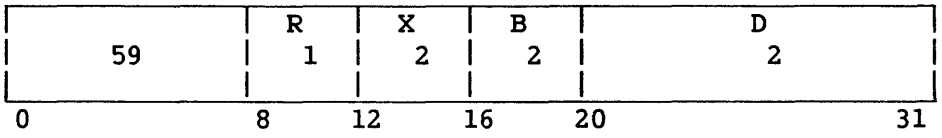
None

COMPARE (CR, C)

CR R1,R2 (RR)



C R1,D2(X2,B2) (RX)



The first operand is compared with the second operand, and the result determines the setting of the condition code. The second operand of the C instruction must be fullword aligned.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code

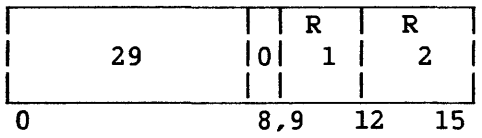
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions

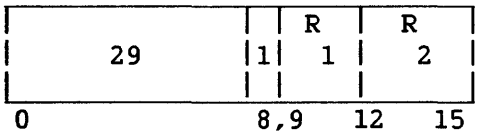
Access (fetch, Operand 2 of C only)
Specification (C only)

COMPARE (FLOATING-POINT) (CDR, CER, CD, CE)

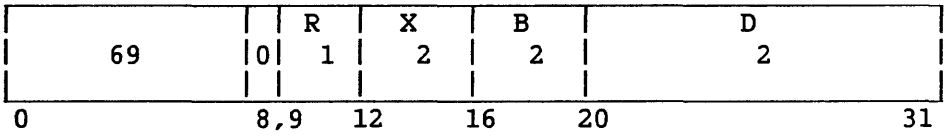
CDR R1,R2 (RR, Long)



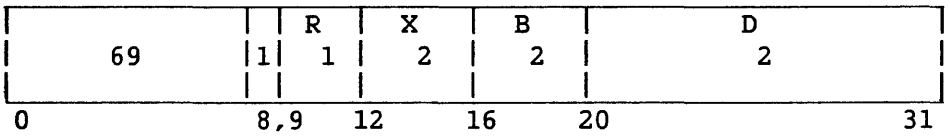
CER R1,R2 (RR, Short)



CD R1,D2(X2,B2) (RX, Long)



CE R1,D2(X2,B2) (RX, Short)



The first operand is compared with the second operand, and the condition code indicates the result.

Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. An exponent inequality is not decisive for magnitude determination, since the fractions may have different numbers of leading 0s. An equality is established by following the rules for normalized floating-point subtraction. When the intermediate sum, including the guard digit, is 0, the operands are equal. Neither operand is changed as a result of the operation.

An exponent-overflow, exponent-underflow, or lost significance exception cannot occur.

Operand 2 of the CD instruction must be fullword aligned.

Resulting Condition Code

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions

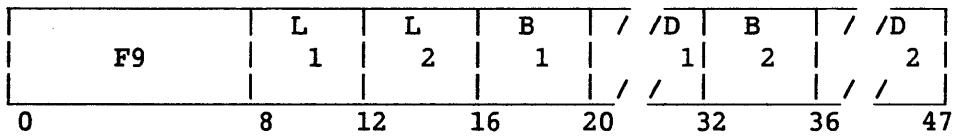
Specification
Access

Programming Note

Condition code 0 (equal comparison) is set when numbers with zero fractions are compared, even when they differ in sign or characteristic.

COMPARE DECIMAL (CP)

CP D1(L1,B1),D2(L2,B2) (SS)



The first operand is compared with the second, and the condition code indicates the comparison result.

Comparison proceeds right to left, taking into account the sign and all digits of both operands. All digits are checked for validity. If the fields are unequal in length, the shorter is extended with 0s on the most significant end. A field with a zero value and positive sign is considered equal to a field with a zero value but negative sign. Neither operand is changed as a result of the operation. Overflow cannot occur in this operation.

The first and second fields may overlap when their least significant bytes coincide. It is possible, therefore, to compare a number to itself.

L1 and L2 are the field lengths in bytes, minus 1.

Resulting Condition Code

0 Operands equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions

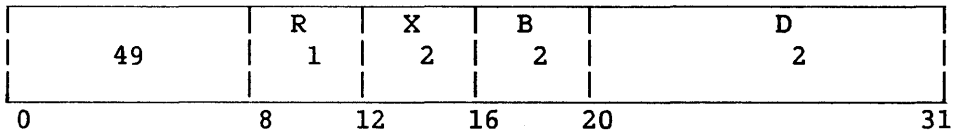
Access (fetch, operands 1 and 2)
Data

Programming Note

The COMPARE DECIMAL instruction is the only COMPARE instruction that processes from right to left, taking signs, 0s, and invalid characters into account, and extending variable-length fields when they are unequal in length.

COMPARE HALFWORD (CH)

```
CH   R1,D2(X2,B2)                (RX)
```



The first operand is compared with the second operand, and the result determines the setting of the condition code. The second operand is two bytes in length, must be halfword aligned, and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the comparison by propagating the sign-bit value through the 16 high-order bit positions.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code

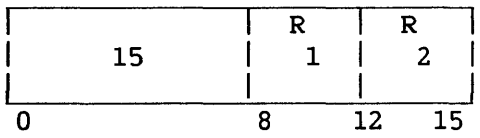
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions

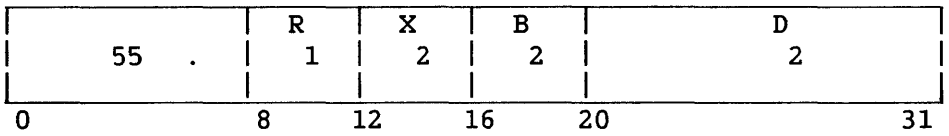
Access (fetch, operand 2)
Specification

COMPARE LOGICAL (CLR, CL, CLI, CLC)

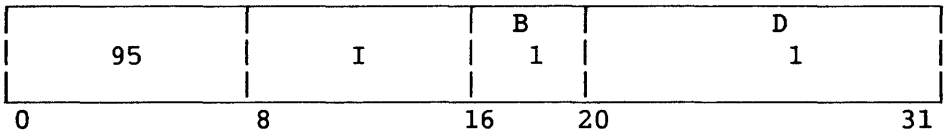
CLR R1,R2 (RR)



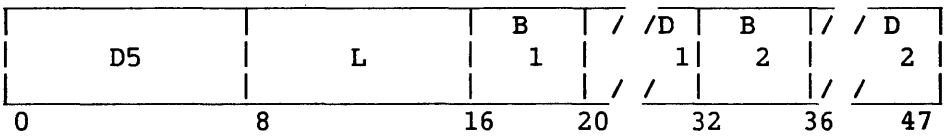
CL R1,D2(X2,B2) (RX)



CLI D1(B1),I2 (SI)



CLC D1(L,B1),D2(B2) (SS)



The first operand is compared with the second operand, and the result is indicated in the condition code. For the CL instruction, the second operand requires fullword alignment.

The instructions allow comparisons that are register-to-register, storage-to-register, instruction-to-storage, and storage-to-storage. The length of the CLC instruction is stored as the actual length minus 1 in the L1 field.

Comparison is unsigned binary, and all codes are valid. The operation proceeds left to right and ends as soon as an inequality is found or the end of the fields is reached. However, when part of an operand in the CLC instruction is specified in an unavailable location, the operation may be terminated by an addressing exception.

Resulting Condition Code

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions

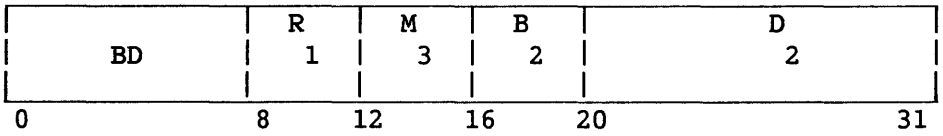
Specification (CL only)
Access (fetch, operand 2, CL and CLC; fetch, operand 1, CLI, CLC)

Programming Note

The COMPARE LOGICAL instructions treat all bits alike as part of an unsigned binary quantity. In variable-length operation, comparison is left to right and may extend to the full specified field length. The operation may be used to compare unsigned packed decimal fields or alphanumeric information in any code that has a collating sequence based on ascending or descending binary values. For example, ASCII has a collating sequence based on ascending binary values.

COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)

CLM R1,M3,D2(B2) (RS)



The second operand is compared with the first operand under control of a mask, and the result is indicated in the condition code.

The contents of the M3 field (bit positions 12-15) are used as a mask. The four bits of the mask, left to right, correspond with the four bytes, left to right, of the general register designated by the R1 field. The byte positions corresponding to 1s in the mask are considered a contiguous field and are compared with the second operand. The second operand is a contiguous field in memory, starting at the second operand address and equal in length to the number of 1s in the mask. The bytes in the general register corresponding to 0s in the mask do not participate in the operation. The comparison is performed with the operands regarded as binary unsigned quantities, with all codes valid. The operation proceeds left to right.

When the mask is not 0, exceptions associated with storage-operand access are recognized only for the number of bytes specified by the mask. However, when part of the designated storage operand is in an inaccessible location but the operation can be completed by using the accessible operand parts, it is unpredictable whether or not the exception for the inaccessible part is indicated. When the mask is 0, access exceptions are recognized for one byte.

Resulting Condition Code

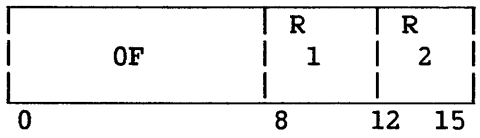
0 Selected bytes are equal, or mask is 0
1 Selected field of first operand is low
2 Selected field of first operand is high
3 --

Program Exceptions

Access (fetch, operand 2)

COMPARE LOGICAL LONG (CLCL)

CLCL R1,R2 (RR)



The first operand is compared with the second operand, and the result is indicated in the condition code.

The R1 and R2 fields each designate an even-odd pair of general registers and must each specify an even-numbered register; otherwise, a specification exception is recognized.

The addresses of the leftmost bytes of the first and second operands, respectively, are specified by Bits 8-31 of general registers R1 and R2. Numbers of bytes in the first and second operands, respectively, are given by Bits 8-31 of general registers R1+1 and R2+1. Bits 0-7 of register R2+1 contain the padding character. Bits 0-7 of registers R1, R1+1, and R2 are ignored.

The comparison is performed with the operands regarded as binary unsigned quantities, with all codes valid. The comparison starts at the high-order end of both fields and proceeds to the right. The operation ends as soon as an inequality is detected or the end of the longer operand is reached. If the operands are not of the same length, the shorter operand is extended with the padding character for purposes of comparison.

If both operands are of zero length, the operands are considered equal.

The execution of the instruction is interruptible. When an interruption occurs after a unit of operation other than the last one, the contents of registers R1+1 and R2+1 are decremented by the number of bytes compared, and the contents of registers R1 and R2 are incremented by the same number, so that the instruction, when re-executed, resumes at the point of interruption. The high-order bytes of registers R1 and R2 are set to 0; the contents of the high-order byte of registers R1+1 and R2+1 remain unchanged. If the operation is interrupted after the shorter operand has been exhausted, the count field pertaining to the shorter operand is 0 and its address is updated accordingly.

The instruction may be refetched from main storage even in the absence of an interruption during execution.

If the operation ends because of a mismatch, the count and address fields at completion identify the byte of mismatch. The contents of bit positions 8-31 of registers R1+1 and R2+1 are decremented by the number of bytes that matched, unless the mismatch occurred with the padding character, in which case the count field for the shorter operand is set to 0. The contents of bit positions 8-31 of registers R1 and R2 are incremented by the amounts by which the corresponding count fields were reduced. If the count fields of both operands are made 0 at completion and the addresses are incremented by the corresponding count values, the contents of bit positions 0-7 of registers R1 and R2 are set to 0, even in the case when one or both of the original count values are 0. The contents of bit positions 0-7 of registers R1+1 and R2+1 remain unchanged.

When part of an operand is designated in an inaccessible location but the operation can be completed by using the available operand parts, it is unpredictable whether an access exception for the inaccessible part is recognized.

When the count field for an operand has the value 0, no access exceptions are recognized for that operand.

Resulting Condition Code:

0 Operands are equal, or both fields have zero length
1 First operand is low
2 First operand is high
3 --

Program Exceptions

Access (fetch, operands 1 and 2)
Specification

Programming Notes

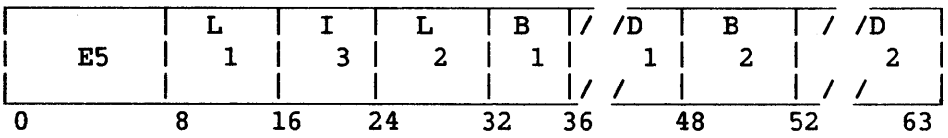
When the contents of the R1 and R2 fields are the same, the condition code is set to 0, but protection and addressing exceptions do not necessarily occur as called for by the operand designation.

Special precautions should be taken when COMPARE LOGICAL LONG is made the subject of EXECUTE. See the programming notes under EXECUTE.

See also the programming notes under MOVE LONG.

COMPARE LOGICAL WITH PAD (CLPC)

CLPC D1(L1,B1),D2(L2,B2),I3 (SSI)



The first operand is compared with the second operand, and the result is indicated in the condition code. Comparison is binary, and all codes are valid. All bits are treated alike as part of an unsigned binary quantity. The operation proceeds left to right and ends as soon as an inequality is found. L1 and L2 are the operand lengths, minus 1. If operand lengths L1 and L2 are unequal, the shorter operand is extended on the right for purposes of comparison by replication of the character specified in the I3 field of the instruction.

The bytes compared are not modified.

Resulting Condition Code

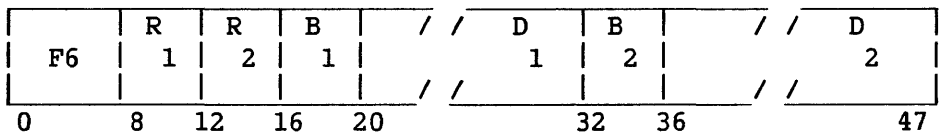
0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Exceptions

Access (fetch, operands 1 and 2)

COMPRESS STRING (COMP)

COMP D1(R1,B1),D2(R2,B2) (SS)



The second operand is placed in the first operand location in a compressed format.

The lengths of operands 1 and 2 are taken from registers R1 and R2, respectively. If the value in either register is 0 or greater than 2048, the instruction terminates immediately with condition code 2, and operand 1 is not changed.

The resulting string in the first operand location contains one or more substrings, each consisting of a length byte followed by one or more data bytes. The length byte format is as follows:

Bit 0 = 0 Uncompressed substring follows
= 1 Compressed substring follows

Bits 1-7 Length of original substring minus 1

A compressed substring is always two bytes in length, and consists of the length byte followed by a byte to be replicated when recreating the original string. All bytes repeated three or more times are compressed; pairs of identical bytes are not compressed.

Resulting Condition Code

0 String successfully compressed; length of compressed string placed in register R1.

1 Compressed string too long for operand 1; register R1 unchanged; data in operand 1 unreliable.

2 Length in R1 or R2 is 0 or greater than 2048; instruction suppressed; register R1 unchanged.

3 --

Program Exceptions

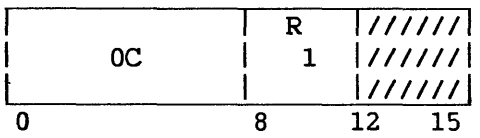
Access (fetch, operand 2; store, operand 1)

Programming Note

Pairs are not compressed. Thus hexadecimal 'AABBCCCCDD' becomes '04AABBCCCCDD' rather than '01AABB81CC00DD'.

CONTROL I/O (CIO)[P]

CIO R1 (RR)



CONTROL I/O causes the addressed device or I/O processor to perform device-dependent or processor-dependent actions. Not all devices and I/O processors accept CIO as a valid request. When issued for a device for which it is not supported, CIO returns a condition code 0, and program execution continues.

Bits 16 to 31 of R1 identify the device. Bits 0 to 15 are ignored. Formats of device addresses for VS systems are described in Section 9.7.

Before the instruction is issued, the command table address (CTA) must be stored in the appropriate entry of the I/O status table, and the I/O command word (IOCW) must be stored in the appropriate entry of the I/O command table (IOCT) addressed by the CTA. These structures are described in Chapter 9.

During instruction execution, the I/O operation defined by the prestored IOCW is presented to the BP, BA, or IOC addressed by the PDA. These processors may accept or reject the command, as explained in Section 9.13. Acceptance of the command is indicated by a zero condition code; rejection, by a non-zero condition code.

It is possible that having been accepted by the addressed BP, BA, or IOC, the I/O command is later rejected because of a condition at the addressed device, or because the addressed device is nonexistent. This rejection is reported in the Status Qualifier Byte (SQB), an extension of the IOSW. The SQB format is described in Section 9.6; types of rejection reported in the SQB are described in Section 9.14.

When an CIO instruction is completed with a condition code of 0, the I/O command is usually carried out; exceptions are noted in the preceding paragraph. A pending I/O interruption is established on completion of the operation. Until the completion interrupt has been received, the IOCW must not be changed. The IOCW is not changed by the I/O processors.

Resulting Condition Code

	VS15, VS65	VS100	VS300
0	Successful	Successful	Successful
1	Not used	Not used	Not used
2	Not used	IOP busy	Not used
3	BP busy	IPC-IN busy	IOC busy or nonexistent

Program Exceptions

Privileged operation

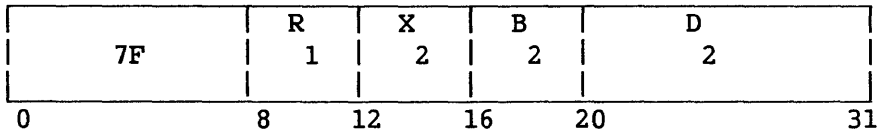
Programming Notes

Telecommunications (TC) IOPs use the SIO instruction rather than CIO for memory diagnostic operations.

When issued for a device for which it is not supported, CIO returns condition code 0, and program execution continues.

CONVERT DECIMAL (FLOATING-POINT) TO PACKED DECIMAL (CVP)

CVP R1,D2(X2,B2) (RX)



The decimal floating-point number in the floating-point register designated by R1 is converted to packed decimal format, and the result is stored in the location specified by the second operand. If the second operand address is not word aligned, a specification exception will occur.

Absolute values greater than 9 99 99 99 99 99 99 99 result in a decimal overflow, and cause a program interruption if the decimal overflow mask bit is 1. In the event of an overflow, the low-order 15 digits plus the sign digit are stored in the second operand.

Resulting Condition Code

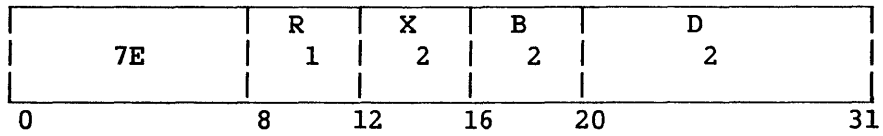
0	Result is 0
1	Result is less than 0
2	Result is greater than 0
3	Overflow

Program Exceptions

Specification
Data
Decimal overflow
Access

CONVERT PACKED DECIMAL TO DECIMAL (FLOATING-POINT) (CVQ)

CVQ R1,D2(X2,B2) (RX)



The 8-byte packed decimal value in the second operand is converted to a normalized decimal floating-point number and placed in the floating-point register designated by R1.

The second operand address must be word aligned, or else a specification exception occurs.

Exponent overflow and exponent underflow cannot occur.

No significance exception will be taken for a zero fraction.

Resulting Condition Code

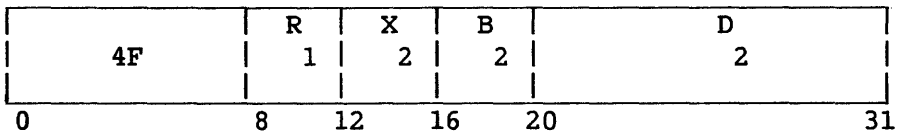
0	Result is 0
1	Result is less than 0
2	Result is greater than 0
3	-

Program Exceptions

Specification
Data
Access

CONVERT TO BINARY (CVB)

```
CVB   R1,D2(X2,B2)              (RX)
```



The radix of the second operand is changed from decimal to binary. The number is treated as a right-aligned signed integer both before and after conversion. The second operand has the packed decimal data format and is checked for valid digit codes. Improper codes cause a program interruption with interruption code 07 (data exception). The decimal operand occupies eight bytes in memory and must be fullword aligned. If the decimal operand is not properly aligned, the instruction will be suppressed and will cause a specification exception. The low-order four bits of the field represent the sign. The remaining 60 bits contain 15 binary-coded-decimal digits in true notation. The result of the conversion is placed in the general register specified by R1. The maximum number that can be converted and still be contained in a 32-bit register is 2,147,483,647; the minimum number is -2,147,483,648. For any decimal number outside this range, the operation is completed by placing the 32 low-order binary bits in the register; a fixed-point divide exception exists, and a program interruption follows. In the case of a negative second operand, the low-order part is in 2's-complement notation.

Resulting Condition Code

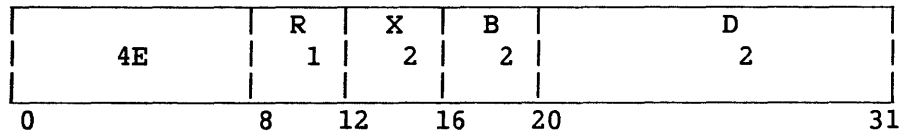
The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2)
Data
Fixed-point divide
Specification

CONVERT TO DECIMAL (CVD)

CVD R1,D2(X2,B2) (RX)



The radix of the first operand is changed from binary to decimal, and the result is stored in the second operand location. The number is treated as a right-aligned signed integer both before and after conversion.

The result is placed in the memory location designated by the second operand and has the packed decimal format. The second operand must occupy eight bytes and must be fullword aligned. If the second operand is not properly aligned, the instruction will be suppressed and will cause a specification exception. A positive sign is encoded as 1111; a negative sign is encoded as 1101. The remaining 60 bits contain 15 binary coded decimal digits in true notation.

Since 15 decimal digits are available for the decimal equivalent of 31 bits, an overflow cannot occur.

Resulting Condition Code

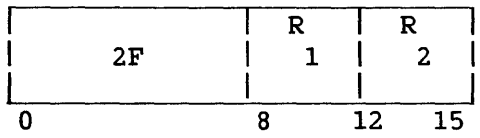
The condition code remains unchanged.

Program Exceptions

Access (store, operand 2)
Specification

CONVERT FLOATING-POINT TO INTEGER (CDI)

CDI R1, R2 (RR)



The integer portion of the floating-point number in the floating-point register designated by the second operand is converted to a binary integer in 2's-complement form, and placed in the general register designated by the first operand. Any binary fraction digits are discarded (right-truncated) in the fixed-point integer result. Values greater than (2**31)-1 or less than -(2**31) result in overflow, and cause a program interruption when the fixed-point overflow mask bit is 1. In the event of overflow, the low-order 32 bits of the correct result are placed in the result register.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 Overflow

Program Exceptions

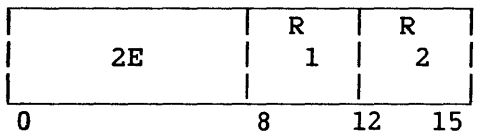
Fixed-point overflow
Specification

Programming Note

To save the fraction before conversion, multiply the floating-point number by that power of 10 corresponding to the degree of precision desired.

CONVERT INTEGER TO FLOATING-POINT (CID)

CID R1, R2 (RR)



The binary integer in the general register designated by the second operand is converted to a normalized floating-point number and placed in the floating-point register designated by the first operand. Exponent overflow and exponent underflow cannot occur. Binary 0 is converted to true floating-point 0.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 --

Program Exceptions

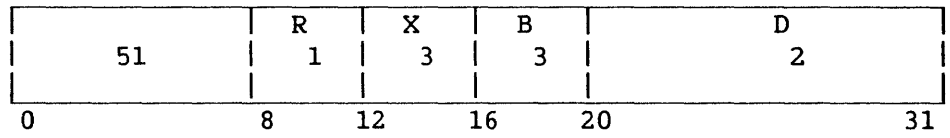
Specification

Programming Note

A significance interrupt will never occur.

DECREMENT AND INSPECT SEMAPHORE (DSEM)

DSEM R1,D2(X3,B3) (RX)



The byte addressed by contents of register R1 is treated as a 2's-complement binary number, and 1 is subtracted from it. If the result is 0 or greater, the next instruction is taken. If the result is less than 0, a binary 1 is added to the high-order byte of the word addressed by the combined second and third operands (D2(X3,B3)), without regard for possible overflow. An enqueuing operation occurs exactly as if the instruction were an ENQ instruction with the same R1, B3, X3, and D2 fields.

If a result of -129 is developed by the subtraction, a fixed-point overflow is indicated. When the fixed-point overflow flag is 1, the exception will be taken.

If there is a fixed-point overflow, the count is updated and the rest of the effects of the instruction are suppressed.

Data fields referenced by this instruction must be aligned as is required for the ENQ instruction. The queue head and queued blocks must not overlap in memory.

Resulting Condition Code

0 Result of subtraction is 0
1 Result of subtraction is less than 0
2 Result of subtraction is greater than 0
3 Overflow

Program Exceptions

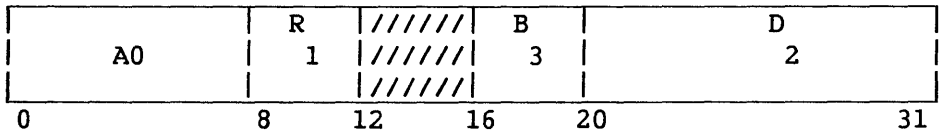
Specification

Fixed-point overflow

Access (fetch and store, operand 1; fetch and store, combined operands 2 and 3 as for ENQ instruction)

DEQUEUE (DEQ)

```
DEQ   R1,D2(B3)                (RS)
```



The first operand addresses a doubleword First-In First-Out (FIFO) queue which consists of a head word and a tail word. When the queue is empty, both the head and the tail words are null (the last 24 bits of each of these words are binary 0s). The dequeuing operation checks for an empty queue first, and if the queue is empty, the third operand is made null and a condition code of 0 is set. If the queue is not empty, the address of the storage block indicated by the head word is placed in the third operand and the chain word at displacement (D2) in the storage block being dequeued is placed in the head word and zeroed in the storage block. If the new head word is null, the queue is empty, and the tail word is also made null. A condition code of 1 is set to indicate that a storage block has been dequeued. The DEQ instruction does not modify or test the first byte of the head or tail pointers or of the chain word.

An addressing exception is recognized and the operation is terminated if the first operand (queue) address is invalid. Both the queue addresses and the chain word location in the dequeued storage block are checked for protection exceptions: the instruction is suppressed if a violation is recognized. A specification exception is recognized and the operation is terminated if one but not both of the head/tail words is null, or if both head/tail words point to the same block but the chain word in the block is not null.

A specification exception is recognized if the head/tail area is not doubleword aligned or if any chain word referenced is not fullword aligned. The queue head and queued blocks must not overlap in memory.

Resulting Condition Code

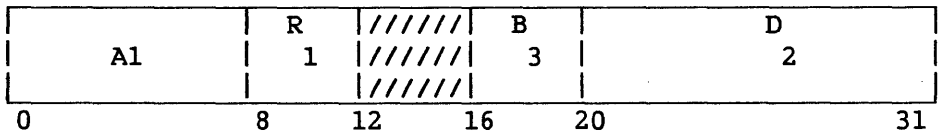
0 No storage blocks queued
1 Storage block dequeued and queue updated
2 --
3 --

Program Exceptions

Access (fetch and store, operand 1 and combined operands 2 and 3)
Specification

DESTACK (DESK)

DESK R1,D2(B3) (RS)



The first operand addresses a LIFO stack pointer to the most recently entered storage block in the stack. When the stack is empty, the stack pointer word is null (the last 24 bits of this word are binary 0s). The third operand defines a register which is to receive the pointer to the destacked storage block, and the second operand defines the displacement of the chain word in the storage blocks in the stack.

The unstacking operation checks for a null stack first, and if the stack is empty, the third operand is made 0 and a condition code of 0 is set. If the stack is not empty, the address of the storage block indicated by the stack pointer word is placed in the third operand, the high-order byte of the third operand is zeroed, the value found in the low-order three bytes of the chain word of the destacked storage block is placed in the low-order three bytes of the stack pointer word, and the chain word is made null. The condition code is set to 1 to indicate that a storage block has been destacked. The DESK instruction does not modify or test the first byte of the stack pointer or chain word.

If the first operand (stack) address is invalid, an addressing exception is recognized, and the operation is terminated. The stack address and the chain word address in the destacked storage block are checked for protection exceptions; the instruction is suppressed if a violation is recognized.

If the stack address is not in a fullword-aligned location, or if any chain word that the instruction references is not fullword aligned, a specification exception is recognized.

Resulting Condition Code

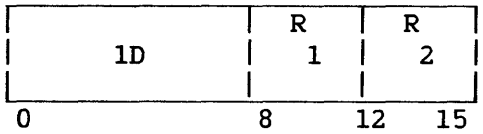
0 No storage blocks stacked
1 Storage block destacked and stack updated
2 --
3 --

Program Exceptions

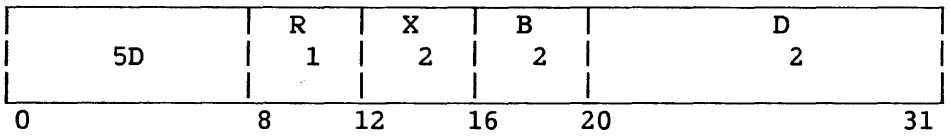
Access (fetch and store, operand 1 and combined operands 2 and 3)
Specification

DIVIDE (DR, D)

DR R1,R2 (RR)



D R1,D2(X2,B2) (RX)



The dividend (first operand) is divided by the divisor (second operand) and replaced by the quotient and remainder.

The dividend is a 64-bit signed integer and occupies the pair of registers beginning with the register specified by the R1 field of the instruction. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in register R1 and the register following R1, respectively. The divisor is a 32-bit signed integer.

The R1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R1 is odd. Operand 2 of the D instruction requires fullword alignment.

The sign of the quotient is determined by the rules of algebra. The remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive. All operands and results are treated as signed integers. When the relative magnitudes of dividend and divisor are such that the quotient cannot be expressed as a 32-bit signed integer, a fixed-point divide exception is recognized (a program interruption occurs, no division takes place, and the dividend remains unchanged in the general registers).

Resulting Condition Code

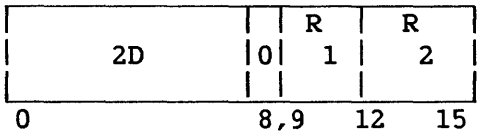
The condition code remains unchanged.

Program Exceptions

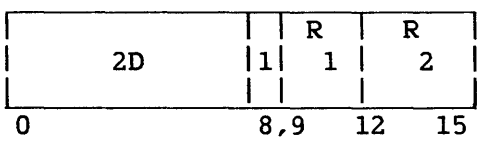
Access (fetch, operand 2 of D only)
Fixed-point divide
Specification

DIVIDE (FLOATING-POINT) (DDR, DER, DD, DE)

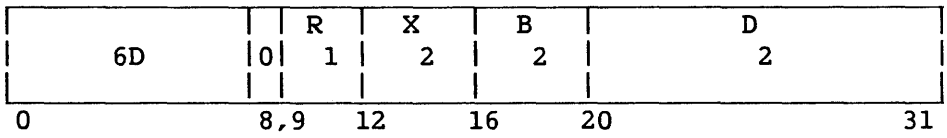
DDR R1,R2 (RR, Long)



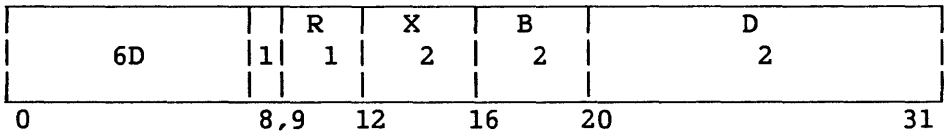
DER R1,R2 (RR, Short)



DD R1,D2(X2,B2) (RX, Long)



DE R1,D2(X2,B2) (RX, Short)



The dividend (the first operand) is divided by the divisor (the second operand) and replaced by the quotient. No remainder is preserved. Operand 2 of the DD and DE instructions must be fullword aligned.

A floating-point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics plus 64 is used as an intermediate quotient characteristic. The sign of the quotient is determined by the rules of algebra.

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right-shift may be called for. The intermediate-quotient characteristic is adjusted for the shifts. All dividend fraction digits participate in forming the quotient, even if the normalized dividend fraction is larger than the normalized divisor fraction. The quotient fraction is truncated to the desired number of digits.

A program interruption for exponent overflow occurs when the final-quotient characteristic exceeds 127. The operation is completed, the fraction is correct and normalized, the sign is correct, and the characteristic is smaller by 128 than the correct characteristic.

If the final quotient characteristic is less than 0 and the exponent underflow mask bit in the PCW is 1, a program interruption for exponent underflow occurs. The fraction is correct and normalized, the sign is correct, and the characteristic is larger by 128 than the correct characteristic. If the corresponding mask bit is not 1, the result is made a true 0. Underflow is not signaled for the intermediate quotient or for the operand characteristics during prenormalization.

When division by a divisor with zero fraction is attempted, the operation is suppressed. The dividend remains unchanged, and a program interruption for floating-point divide occurs. When the dividend fraction is 0, the quotient fraction will be 0. The quotient sign and characteristic are made 0, yielding a true zero result without taking the program interruption for exponent underflow and exponent overflow. The program interruption for significance is never taken for division.

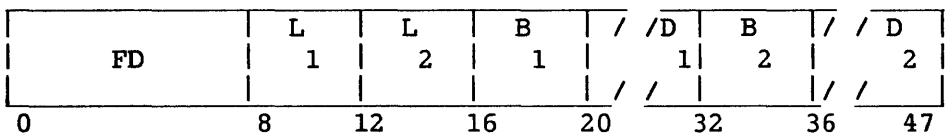
Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Specification
Exponent overflow
Exponent underflow
Floating-point divide
Access

DP D1(L1,B1),D2(L2,B2) (SS)



The dividend (the first operand) is divided by the divisor (the second operand) and replaced by the quotient and remainder.

The quotient field is placed leftmost in the first operand field. The remainder field is placed rightmost in the first operand field and has a size equal to the divisor size. Together, the quotient and remainder occupy the entire dividend field; therefore, the address of the quotient field is the address of the first operand. L1 and L2 are the field lengths in bytes, minus 1. The size of the quotient field in bytes is L1 - L2. When the divisor length code (L2) is larger than 7 (15 digits and sign) or is greater than or equal to the dividend length code (L1), a specification exception is recognized. The operation is suppressed, and a program interruption occurs.

If division by 0 is attempted, a decimal divide exception is recognized and the operation is terminated.

The dividend, divisor, quotient, and remainder are all signed integers, right-aligned in their fields. The sign of the quotient is determined by the rules of algebra from dividend and divisor signs. The sign of the remainder has the same value as the dividend sign.

Division is algebraic, taking into account the sign and all digits of both operands. All digits are checked for validity. If necessary, 0s are supplied for either operand on the most significant end. When the first operand field (L1) is too short to contain all significant digits of the quotient, the operation is terminated and the overflow condition is set.

A quotient larger than the number of digits allowed is recognized as a decimal divide exception. The operation is terminated.

The divisor and dividend fields may overlap if their least significant bytes coincide.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Access (store operand 1, fetch operand 2)
Data
Decimal divide
Specification

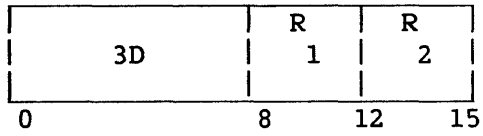
Programming Notes

The maximum dividend size is 31 digits plus sign. Since the smallest remainder size is 1 digit and sign, the maximum quotient size is 29 digits and sign.

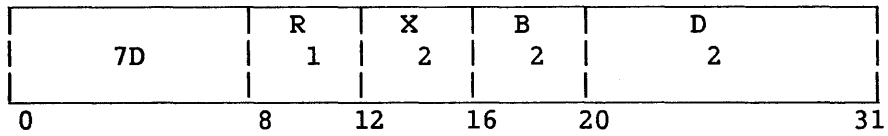
The condition for an overflow exception can be determined by a trial subtraction. The leftmost digit of the divisor field is aligned with the second-to-leftmost digit of the dividend field. When the divisor, so aligned, is less than or equal to the dividend, an overflow exception is indicated.

DIVIDE DECIMAL (FLOATING-POINT) (DQR, DQ)

DQR R1,R2 (RR)



DQ R1,D2(X2,B2) (RX)



The dividend (the first operand) is divided by the divisor (the second operand), and the quotient replaces the first operand. Any remainder is discarded. Fullword alignment is required.

A decimal floating-point division consists of a characteristic subtraction and a fraction division. The difference between the dividend and divisor characteristics plus 64 is used as an intermediate quotient characteristic. The sign of the quotient is determined by the rules of algebra.

The quotient fraction is normalized by prenormalizing the operands. Postnormalizing the intermediate quotient is never necessary, but a right-shift may be called for. The intermediate-quotient characteristic is adjusted for the shifts. The quotient fraction is truncated to 14 digits.

A program interruption for exponent overflow occurs when the final-quotient characteristic exceeds 127. The operation is completed, the fraction is correct and normalized, the sign is correct, and the characteristic is 128 smaller than the correct characteristic.

If the final quotient characteristic is less than zero and the exponent underflow mask bit is 1, a program interruption for exponent underflow occurs. The fraction is correct and normalized, the sign is correct, and the characteristic is 128 larger than the correct characteristic. If the corresponding mask bit is 0, the result is made a true zero. Underflow cannot occur during prenormalization.

When division by a divisor with zero fraction is attempted, the operation is suppressed. The dividend remains unchanged, and a program interruption for floating-point divide occurs. When the dividend fraction is zero, the quotient result is made a true zero without taking a program interruption (for exponent underflow or overflow). The program interruption for significance is never taken for division.

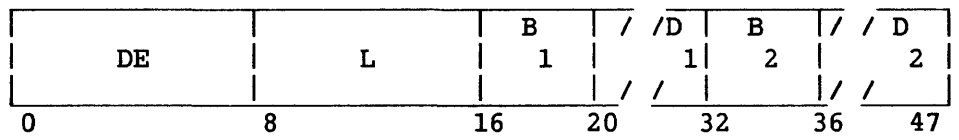
Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Specification
Data
Exponent overflow
Exponent underflow
Access (DQ only)
Floating-point divide

ED D1(L,B1),D2(B2) (SS)



The format of the source (the second operand) is changed from packed to zoned, and is modified under control of the pattern (the first operand). The edited result replaces the pattern.

Editing includes sign and punctuation control, and the suppressing and protecting of leading 0s. It also facilitates programmed blanking of all-zero fields. Numeric information in the source may be interspersed with text from the pattern.

The length field applies to the pattern (the first operand). L is equal to the pattern length minus 1. The pattern has the zoned format and may contain any character. The source (the second operand) has the packed format. The leftmost four bits of a source byte must specify a decimal digit code (0000-1001); a code of 1010-1111 is recognized as a data exception and causes a program interruption. The rightmost four bits may specify either a sign or a decimal digit. Overlapping pattern and source fields give unpredictable results.

During the editing process, each character of the pattern is affected in one of three ways:

1. It is left unchanged.

2. It is replaced by a source digit expanded to zoned format.

3. It is replaced by the first character in the pattern, called the fill character.

Which of the three actions takes place is determined by one or more of the following: the state of the significance indicator, the type of the pattern character, and whether the source digit examined is 0.

Significance Indicator

The significance indicator is a bit that by its state (on or off) indicates the significance or nonsignificance, respectively, of subsequent source digits or message characters. Significant source digits replace their corresponding digit selectors or significance starters in the result. Significant message characters remain unchanged in the result.

The significance indicator also indicates the negative (on) or positive (off) value of the source, and is used as one factor in the setting of the condition code. The total number of digit selectors, significance starters, and immediate significance starters in the pattern must equal the number of source digits to be edited; otherwise, the state of the significance indicator and the condition code at the end of instruction execution is unspecified.

The indicator is set to the off state if it is not already so set, either at the start of the editing operation or when the decimal digit in the last byte of the source exhausts the digit selectors and significance starters of the pattern and the low-order part of the same byte does not contain 1101.

The indicator is set to the on state, if it is not already so set, when a significance starter or immediate significance starter is encountered whose source digit is a valid decimal digit, or when a digit selector is encountered whose source digit is a nonzero decimal digit, provided in either instance that the low order source byte does not have a plus code in the four low-order bit positions.

In all other situations, the indicator is not changed. A minus sign code has no effect on the significance indicator.

Pattern Characters

There are five types of pattern characters: fill characters, digit selectors, significance starters, immediate significance starters, and message characters. Their coding is presented in Table 8-1.

Table 8-1. Pattern Character Coding

Pattern Character	Binary Code	Hexadecimal Code
Fill character	Any	Any
Digit selector	0001 0000	10
Significance starter	0001 0001	11
Immediate significance starter	0001 0010	12
Message character	Any other	Any other

The fill character is the first character of the pattern. It may have any code and may concurrently specify a control function. If this character is a digit selector, significance starter, or immediate significance starter, the indicated editing action is taken after the code has been assigned to the fill character.

The digit selector makes source digits appear as in the source field, unless the significance indicator is off.

The significance starter functions as a digit selector, except that it turns on the significance indicator after processing its corresponding source digit.

An immediate significance starter functions as a significance starter, except that it turns on the significance indicator before processing its source digit.

Message characters in the pattern are replaced by the fill character, or they remain unchanged in the result, depending on the state of the significance indicator. They may thus be used for padding, punctuation, or text in the significant portion of a field or for the insertion of sign-dependent symbols.

The Edit Operation

The detection of a digit selector, significance starter, or immediate significance starter in the pattern causes an examination to be made of the significance indicator and of a source digit. As a result, either the expanded source digit or the fill character, as appropriate, is selected to replace the pattern character. Additionally, encountering a digit selector, significance starter, or immediate significance starter may cause the significance indicator to be changed.

Each time a digit selector, significance starter, or immediate significance starter is encountered in the pattern, a new source digit is examined for placement in the pattern field. The source digit either is zoned and replaces the pattern character or is disregarded. When a code that is not between 0000 and 1001 is detected in the four high-order bit positions, the operation is terminated with a data exception.

The source digits are selected one byte at a time, and a source byte is fetched for inspection only once during an editing operation. Each source digit is examined once and only once for a zero value. The leftmost four bits of each byte are examined first, and the rightmost four bits, when they represent a decimal-digit code, remain available for the next pattern character that calls for a digit examination. Source digits are examined until the digit selectors, significance starters, and immediate significance starters of the pattern are exhausted. If more than 32 digits must be examined, or if a source digit with codes 1010 through 1111 is examined in response to a digit selector, significance starter, or immediate significance starter, the operation is terminated with a data exception.

When the source digit is stored in the result, its code is expanded from the packed to the zoned format by attaching the zone code 0011.

The field resulting from an editing operation replaces and is equal in length to the pattern. It is composed of pattern characters, fill characters, and zoned source digits.

If the pattern character is a message character and the significance indicator is on, the message character remains unchanged in the result. If the significance indicator is off when a message character is encountered in the pattern, the fill character replaces the pattern character in the result.

If a digit selector or significance starter is encountered in the pattern when the significance indicator is off and the source digit is 0, the source digit is considered nonsignificant, and the fill character replaces the pattern character. If an immediate significance starter is encountered in the pattern with the significance indicator off and the source digit 0, the source digit is considered significant, is zoned, and replaces the pattern character in the result. If a digit selector, significance starter, or immediate significance starter is encountered either with the significance indicator on or with a nonzero decimal source digit, the source digit is considered significant, is zoned, and replaces the pattern character in the result.

Result Conditions. All digits examined are tested for the code 0000. The sign of the field edited and whether all source digits in the field contain 0s are recorded in the condition code at the completion of the editing operation.

The condition code is made 0 when the field is 0, that is, when all source digits examined are 0s. When the pattern has no digit selectors or significance starters, the source is not examined, and the condition code is made 0.

When the editid field is nonzero and the significance indicator is on, the condition code is made 1 to indicate a result field less than 0.

When the edited field is nonzero and the significance indicator is off, the condition code is made 2 to indicate a result field is greater than 0.

Summary. Table 8-2 summarizes the functions of the editing operation. The leftmost four columns list all the significant combinations of the four conditions that can be encountered in the execution of an editing operation. The two rightmost columns list the action taken for each case -- the type of character placed in the result field and the new setting of the significance indicator.

The last column (column 6) of the table shows the state of significance indicator at the end of digit examination when the examined source digit (column 3) is the high-order digit in its byte. Column 4 refers to the low-order source digit in the same byte as the examined source digit. To determine the state of the significance indicator at the end of digit examination when the examined digit is the low-order digit in its byte, refer to the entry in column 6 on the same row which shows the value of the examined digit (column 3) and the value "No" (column 4).

Table 8-2. Summary of Editing Operation

Conditions				Results	
Pattern Character	Previous State of Significance Indicator	Source Digit	Low-Order Source Digit Is a Plus Sign	Result Character	State of Significance Indicator at End of Digit Examination
Digit selector	Off	0	*	Fill character	Off
		1-9	No	Source digit	On
		1-9	Yes	Source digit	Off
	On	0-9	No	Source digit	On
		0-9	Yes	Source digit	Off
Significance starter	Off	0	No	Fill character	On
		0	Yes	Fill character	Off
		1-9	No	Source digit	On
		1-9	Yes	Source digit	Off
	On	0-9	No	Source digit	On
		0-9	Yes	Source digit	Off
Immediate significance starter	Off	0-9	No	Source digit	On
		0-9	Yes	Source digit	Off
	On	0-9	No	Source digit	On
		0-9	Yes	Source digit	Off
Message character	Off	**	**	Fill character	Off
	On	**	**	Message character	On

* No effect on significance indicator.
** Not applicable because source digit not examined.

Resulting Condition Code

0 Field is 0
1 Field is less than 0 (significance indicator on)
2 Field is greater than 0 (significance indicator off)
3 --

Program Exceptions

Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Notes

As a rule, the source is shorter than the pattern because for each source digit a zone and numeric are inserted in the result.

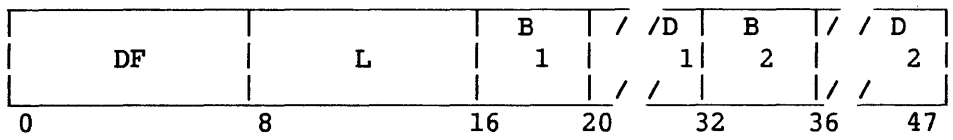
If the fill character is a blank, if no significance starter or immediate significance starter appears in the pattern, and if the source is all 0s, the editing operation blanks the result field.

The resultant condition code indicates whether or not the field is all 0s, and, if the code is not 0, reflects the state of the significance indicator. The significance indicator reflects the sign of the source field only if the last source digit examined is in a high-order digit position.

Address translation demands that operands be located in main memory before instruction execution may begin, so the length of operand 2 must be determined before execution. This is accomplished by scanning operand 1 for significance starters and digit selectors; the total number divided by 2 (and rounded up if total number is odd) yields the number of bytes in operand 2.

EDIT AND MARK (EDMK)

EDMK D1(L,B1),D2(B2) (SS)



The format of the source (the second operand) is changed from packed to zoned and is modified under control of the pattern (the first operand).

The address of the first significant result character is recorded in General Register 1. The edited result replaces the pattern.

The instruction EDIT AND MARK is identical to EDIT, but it has the additional function of inserting the address of the result character in Bits 8-31 of General Register 1 whenever the result character is a zoned source digit and the significance indicator was off before the examination. The use of General Register 1 is implied. The contents of Bits 0-7 of the register are not changed.

Resulting Condition Code

0 Last field is 0
1 Last field is less than 0 (significance indicator on)
2 Last field is greater than 0 (significance indicator off)
3 --

Program Exceptions

Access (fetch, operand 2; fetch and store, operand 1)
Data

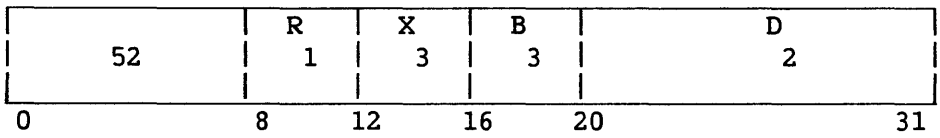
Programming Notes

The instruction EDIT AND MARK facilitates the programming of floating currency-symbol insertion. The character address inserted in General Register 1 is 1 more than the address where a floating currency sign would be inserted. The instruction BRANCH ON COUNT (BCTR), with 0 in the R2 field, may be used to reduce the inserted address by 1.

The character address is not stored when significance is forced. To ensure that General Register 1 contains a valid address when significance is forced, it is necessary to place in the register beforehand the address of the pattern character that immediately follows the significance starter.

ENQUEUE (ENQ)

ENQ R1,D2(X3,B3) (RX)



A storage block beginning at the location specified by the third operand (the sum of the contents of registers B3 and X3) is enqueued on the First-In First-Out (FIFO) queue specified by the first operand. The queue head addressed by register R1 is composed of two successive words of storage of which the first word, or head word, is a pointer to the first storage block in the queue, and the second word, or tail word, is a pointer to the last storage block in the queue. When the FIFO queue is empty, both the head and the tail pointers are null (the last 24 bits of each of these words are binary 0s). The head pointer must be doubleword aligned.

When a storage block is queued, the head and tail pointers are checked, and if they are null, the third operand address is placed in both the head and tail queue positions. If the queue pointers are both not null, then the third operand address is placed in the block pointed to by the tail pointer at the displacement position specified by the second operand. The third operand is then placed in the tail queue position, and in all cases, the chain word in the queued storage block is made null. Thus, blocks are enqueued so that the word at the chain word displacement in each block points to the first location of the next block in the queue, and the last block in the queue has a null chain word. ENQ does not test or change the first byte of either the head or tail pointer or the chain words.

An addressing exception is recognized, and the operation terminated, if either the first-operand (queue) address or the third-operand (storage block) address is invalid. Both the queue addresses and the chain word locations in any storage blocks that are modified in the queue are checked for protection boundary violations and for modification trap exceptions. The instruction is suppressed if a violation occurs. A specification exception is recognized and the operation is terminated if one but not both of the head/tail words is null. The head/tail pointer must be doubleword aligned and the chain words must be fullword aligned or a specification exception will be recognized. The queue head and queued blocks must not overlap in memory.

Resulting Condition Code

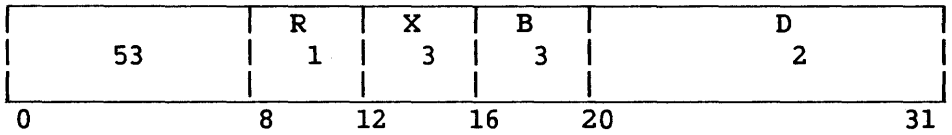
The condition code remains unchanged.

Program Exceptions

Access (fetch and store, operand 1 and combined operands 2 and 3)
Specification

ENSTACK (ENSK)

ENSK R1,D2(X3,B3) (RX)



A storage block beginning at the location specified by the third operand (the sum of the contents of registers B3 and X3) is stacked in the Last-In First-Out (LIFO) stack specified by the first operand. The stack head addressed by register R1 consists of one aligned word of storage that is a stack pointer to the last (or most recent) storage block placed into the stack. When the stack is empty, the stack pointer is null (the last 24 bits of this word are binary 0s). The second operand is a displacement from the start of the storage block to the chain word in the storage block.

When a storage block is stacked, the stack pointer word is placed in the chain word of the storage block being stacked, and the pointer to the start of the storage block (third operand value) is placed in the stack pointer word. Storage blocks are intended for removal from a LIFO stack in the reverse order from the sequence in which they were added to the stack, since the only pointer kept for a LIFO stack is to the last stacked storage block. ENSK does not change or test the first byte of the stack pointer or the chain word.

An addressing exception is recognized and the operation terminated if either the first operand (stack) address or the third operand (storage block) address is invalid. Both the stack address and the chain word location in the storage block are checked for protection violations and for modification trap exceptions; the instruction is suppressed if a violation occurs. Both the stack pointer and the chain word must be fullword aligned.

Resulting Condition Code

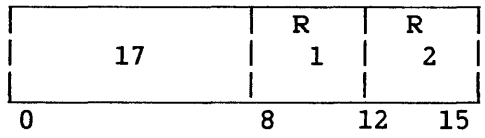
The condition code remains unchanged.

Program Exceptions

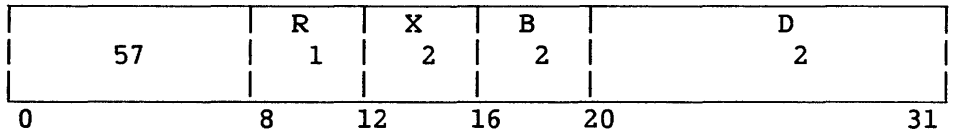
Access (fetch and store, operand 1 and combined operands 2 and 3)
Specification

EXCLUSIVE OR (XR, X, XI, XC)

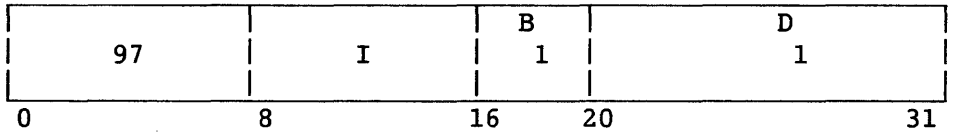
XR R1,R2 (RR)



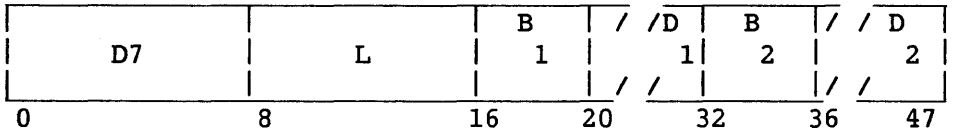
X R1,D2(X2,B2) (RX)



XI D1(B1),I2 (SI)



XC D1(L,B1),D2(B2) (SS)



The modulo-two sum ("exclusive OR") of the bits of the first and second operand is placed in the first operand location.

Operands are treated as unstructured logical quantities, and the connective EXCLUSIVE OR is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to 0.

The instruction differs from AND and OR only in the connective applied.

Operand 2 of the X instruction requires fullword alignment. For the XC instruction, L is the length of each operand, minus 1.

Resulting Condition Code

0 Result is 0
1 Result not 0
2 --
3 --

Program Exceptions

Access (fetch, operand 2, X and XC; fetch and store, operand 1, XI and XC)

Specification

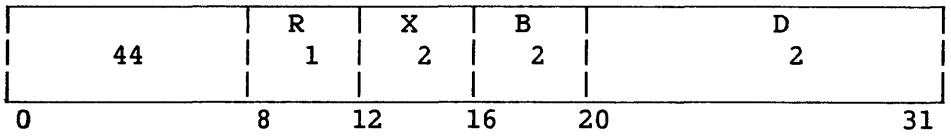
Programming Notes

The EXCLUSIVE OR instruction may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.

Any field EXCLUSIVE ORed with itself becomes all 0s.

EXECUTE (EX)

```
EX   R1,D2(X2,B2)                (RX)
```



Bits 8-15 of the instruction designated by the second-operand address are ORed with Bits 24-31 of the register specified by R1, except when Register 0 is specified, which indicates that no modification takes place. The resulting subject instruction is then executed. The subject instruction may be two, four, six, or eight bytes in length.

The ORing does not change either the contents of the register specified by R1 or the instruction in memory, and it is effective only for the interpretation of the instruction to be executed. The execution and exception handling of the subject instruction are exactly as if the subject instruction were obtained in normal sequential operation, except for the instruction address. The instruction address of the current PCW is increased by the length of EXECUTE. This updated address of EXECUTE is used as part of the link information when the subject instruction is BRANCH AND LINK. When the subject instruction is a successful branching instruction, the updated instruction address of the current PCW is replaced by the branch address specified by the subject instruction.

When the subject instruction is in turn an EXECUTE, an execute exception is recognized, and the operation is suppressed. The subject instruction must be halfword aligned; otherwise, a specification exception is recognized.

Resulting Condition Code

The condition code may be set by the subject instruction.

Program Exceptions

Execute
Access (fetch, operand 2)
Specification

Programming Notes

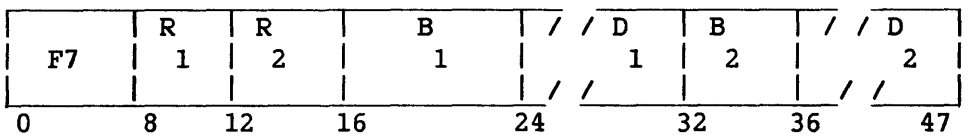
The ORing of eight bits from the general register with the designated instruction permits indirect length, index, mask, immediate data, and arithmetic-register specification. An addressing or specification exception may be caused by EXECUTE or by the subject instruction.

When an interruptible instruction is made a target of EXECUTE, the program usually should not specify any register updated by the interruptible instruction as the R1, X2, or B2 register of the EXECUTE, since if the instruction is refetched, the updated values of these registers will be used in execution of the EXECUTE. Similarly, the program should not let the destination field of an MVCL instruction include the location of the EXECUTE.

When a relative branch instruction is the target of EXECUTE, the branch address is relative to the EXECUTE and not to the target instruction.

EXPAND STRING (XPAND)

XPAND D1(R1,B1),D2(R2,B2) (SS)



The second operand, assumed to be a character string compressed by the COMP instruction, is placed in the first operand location in expanded form.

The lengths of operands 1 and 2 are taken from registers R1 and R2, respectively. If the value in either register is 0 or greater than 2048, the instruction terminates immediately with condition code 2, and operand 1 is unchanged.

The source string is interpreted as a concatenation of subfields, each beginning with a length byte in the following form:

Bit 0 = 0 Uncompressed substring follows

Bit 0 = 1 Following byte to be replicated as many times as specified

Bits 1-7 Length of expanded substring minus 1

Resulting Condition Code

0 String successfully expanded; length of expanded string placed in register R1

1 Expanded string too long for operand 1; register R1 unchanged; data in operand 1 valid

2 Length in R1 or R2 equal to 0 or greater than 2048; instruction suppressed; register R1 unchanged

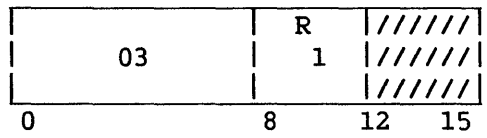
3 Length byte in operand 2 indicates that a source subfield extends beyond the source area defined by the R2 length; the instruction terminates; operand 1 data and register R1 contents are unreliable.

Program Exceptions

Access (fetch, operand 2; store, operand 1)

HALT I/O (HIO)[P]

HIO R1 (RR)



HALT I/O causes the addressed device to terminate the current operation, if any. HALT I/O is executed only when the system is in the supervisor state. I/O interrupts should be disabled.

Bits 16 to 31 of R1 identify the device address. Bits 0 to 15 are ignored.

When the HALT I/O instruction is issued to an active I/O device, the I/O operation may be terminated before all data specified in the operation has been transferred, or before the operation at the device has reached its normal ending point. A completion interruption becomes pending when the I/O operation has been terminated. The associated IOSW shows normal completion and, after the halt of a data transfer operation, a nonzero residual byte count.

If the HIO instruction receives an IOP BUSY indication, the HIO was not accepted. This also indicates that an IOP NOW READY interrupt will be made pending. (The IOP BUSY condition and IOP NOW READY interrupt are encountered on the VS100 system only.)

The meaning of the condition codes returned for this instruction is explained in Section 9.13.

Resulting Condition Code

	VS15, VS65	VS100	VS300
0	Successful	Successful	Successful
1	Not used	Not used	Not used
2	Not used	IOP busy	Not used
3	BP busy	IPC-IN busy	IOC busy or non-existent

Program Exceptions

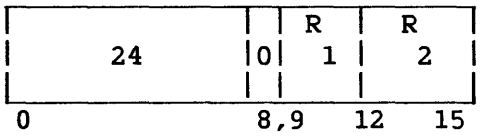
Privileged operation

Programming Notes

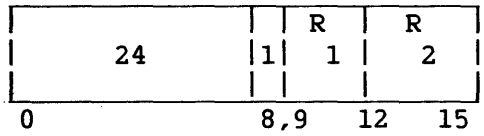
Not all devices support HALT I/O. When it is issued for a device for which it is not appropriate, the HIO instruction returns condition code 0, and program execution continues.

HALVE (FLOATING-POINT) (HDR, HER)

HDR R1,R2 (RR, Long)



HER R1,R2 (RR, Short)



The second operand is divided by 2, and the normalized quotient is placed in the first operand location. The second operand remains unchanged.

The fraction of the second operand is shifted right one bit position, placing the contents of the low-order bit position in the high-order bit position of the guard digit and introducing a 0 into the high-order bit position of the fraction. The intermediate result is subsequently normalized, and the normalized quotient is placed in the first operand location. The guard digit participates in the normalization.

When normalization causes the characteristic to become less than zero, exponent underflow occurs. If the exponent underflow mask in the PCW is 0, the sign, characteristic, and fraction are set to 0, thus making the result a true 0. If the exponent underflow mask is 1, a program interruption occurs. The result is normalized, its sign and fraction remain correct, and the characteristic is made 128 larger than the correct characteristic.

When the fraction of the second operand is 0, the sign, characteristic, and fraction of the result are made 0. No normalization is attempted, and a significance exception is not recognized.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Specification
Exponent underflow

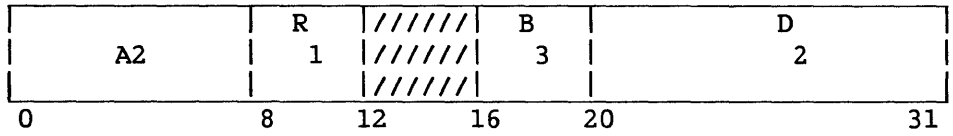
Programming Notes

The HALVE operation is identical to a divide operation with the number 2 as divisor, or to a multiply operation with 1/2 as a multiplier.

The result of HALVE is replaced by a true 0 only when the second operand fraction is 0, or when exponent underflow occurs with the exponent-underflow mask set to 0. When the fraction of the second operand is 0 except for the low-order bit position, the low-order 1 is shifted into the guard digit position and participates in the postnormalization.

INCREMENT AND INSPECT SEMAPHORE (ISEM)

```
ISEM   R1,D2(B3)                (RS)
```



The byte addressed by contents of register R1 is treated as a 2's-complement binary number, and 1 is added to it. If the result is greater than 0, the next instruction is taken. If the result is less than or equal to 0, a dequeuing operation occurs exactly as if the instruction were a DEQ instruction with the same R1, B3, and D2 fields, and a binary 1 is subtracted from the byte at displacement D2 in the dequeued block, without regard for possible overflow. If a result of 128 is developed, a fixed-point overflow is indicated. When the fixed-point overflow flag is 1, the exception will be taken. If there is a fixed-point overflow, the count is updated and the other effects of the instruction are suppressed. Overflows in the chain field will not cause an overflow indication or a program check.

Data fields referenced by this instruction must be aligned as required for the DEQ instruction. The queue head and queued blocks must not overlap in memory.

Resulting Condition Code

0 Result of addition not greater than 0, no block dequeued
1 Result of addition not greater than 0, block dequeued
2 Result of addition greater than 0
3 Overflow

Program Exceptions

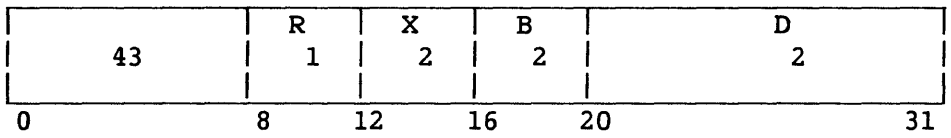
Specification

Fixed-point overflow

Access (fetch and store, operand 1; fetch and store, operands 2 and 3 as for DEQ instruction)

INSERT CHARACTER (IC)

IC R1,D2(X2,B2) (RX)



The 8-bit character at the second operand address is inserted into the low-order byte of the register specified as the first operand location. The remaining bits of the register remain unchanged.

IC is a storage-to-general-register instruction. The byte to be inserted is not changed or inspected.

Resulting Condition Code

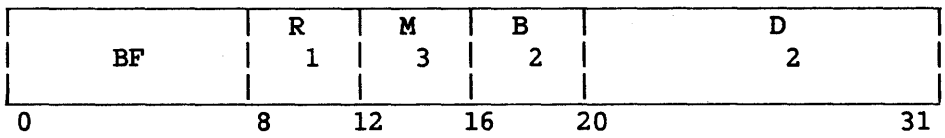
The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2)

INSERT CHARACTERS UNDER MASK (ICM)

ICM R1,M3,D2(B2) (RS)



Bytes from contiguous locations beginning at the second operand address are inserted into the first operand location under control of a mask.

The contents of the M3 field, Bits 12-15, are used as a mask. The four bits of the mask, left to right, correspond with the four bytes, left to right, of the general register designated by the R1 field. The byte positions corresponding to 1s in the mask are filled, in the order of ascending byte numbers, with bytes from the storage operand. Bytes are fetched from contiguous memory locations beginning at the second operand address. The length of the second operand is equal to the number of 1s in the mask. The bytes in the general register corresponding to 0s in the mask remain unchanged.

The resulting condition code is based on the mask and on the value of the bits inserted. When the mask is 0 or when all inserted bits are 0, the condition code is made 0. When not all inserted bits are 0, the code is set according to the leftmost bit of the storage operand: if this bit is 1, the code is made 1 to indicate a negative algebraic value; if this bit is 0, the code is set to 2, reflecting a positive algebraic value. When the mask is not 0, exceptions associated with storage operand access are recognized only for the number of bytes specified by the mask. When the mask is 0, access exceptions are recognized for one byte.

Resulting Condition Code

0 All inserted bits are 0s, or mask is 0
1 First bit of the inserted field is 1
2 First bit of the inserted field is 0 and not all inserted bits are 0s
3 --

Program Exceptions

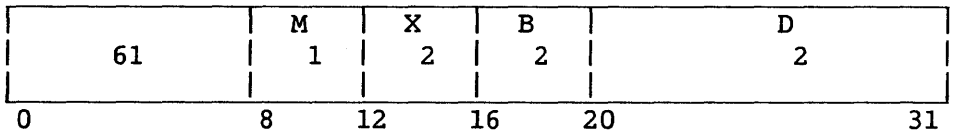
Access (fetch, operand 2)

Programming Note

The condition code for INSERT CHARACTERS UNDER MASK is defined such that when the mask is 1111, the instruction causes the same condition code to be set as for LOAD AND TEST.

JUMP TO SUBROUTINE ON CONDITION INDIRECT (JSCI)

JSCI M1,D2(X2,B2) (RX)



The updated instruction address is replaced by a branch address if the state of the condition code is as specified by M1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The second operand must be fullword aligned. It addresses a word operand in memory. In the case of a branch, the three low-order bytes of the word operand are compared with the three low-order bytes of Control Registers 6 and 7. (These bytes in Control Registers 6 and 7 indicate, respectively, the start and end of a Linkage Table.)

If the three low-order bytes of the word operand fall outside the range of addresses in Control Registers 6 and 7, those bytes are used as the branch address, and the following actions occur: The PCW and the updated instruction address are pushed onto the current system stack. Then pushed onto the stack is a byte consisting of four high-order zeros, followed by the three PCW process level bits, followed by a binary 0, which indicates a JSCI-type save area. Then the three low-order bytes of Control Register 1 are pushed onto the stack, followed by the entire contents of General Registers 14 to 0. After these items have been pushed onto the stack and the stack pointer (Register 15) has been updated, the value in Control Register 1 is set to the current value of the stack pointer, with a high-order byte of binary 0s. Control then passes to the branch address. (Figure 4-11 in Chapter 4 of this manual shows the format of data pushed onto the stack by execution of JSCI.)

Alternatively, if the three low-order bytes of the word operand fall within the range of addresses in Control Registers 6 and 7, those bytes are used as the address of a Linkage Table entry whose second, third, and fourth bytes specify the branch address. The process level bits of the PCW are compared with the execution process level bits of the Linkage Table Entry, and one of the following actions then occurs:

1. If the value of the execution level field is greater than that of the process level field, a stack switch takes place: The contents of General Register 15, Control Register 2, and Control Register 1 are saved in the Stack Header Block addressed by Control Register 8. The value of the execution level field is used to form an offset into the Stack Header Block Table. The address at this offset is loaded into Control Register 8. General register 15 is loaded with the contents of the first four bytes of the Stack Header Block addressed by Control Register 8; Control Register 2 is loaded with the contents of the next four bytes in the same SHB. The three low order bytes of General Register 15 and of Control Register 2 now address, respectively, the top and limit of the active system stack.

   Status information is pushed onto the stack, as already described. Then the three low order bytes of the Linkage Table Entry, if non-zero, are loaded into General Register 14; these bytes address the static area of the called subroutine. The process level bits of the PCW are set equal to the execution process level bits in the Linkage Table entry. Control then passes to the branch address.

2. If the value of the execution level field is less than or equal to that of the process level field, no stack switch takes place. The three low order bytes of the Linkage Table Entry, if non-zero, are loaded into General Register 14; these bytes address the static area of the called subroutine. Status information is pushed onto the active system stack, whose lowest byte location is addressed by General Register 15. Control then passes to the branch address.

The M1 field is used as a 4-bit mask. The four bits of the mask correspond, left to right, with the four condition codes as follows:

Instruction Bit	Mask Position Value	Condition Code
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the condition code has a corresponding mask bit of 1.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Stack overflow

Access (fetch, operand 2; store, the bytes pushed onto the stack if the branch is taken)

Specification

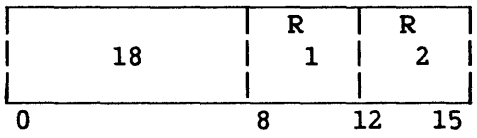
Programming Note

This instruction is a conditional indirect branch. If the branch is taken, status will be saved on a stack that will allow the RTC instruction to return control to the location after the JSCI instruction. Control Register 1 is used to point to the status information saved by a previously executed JSCI instruction.

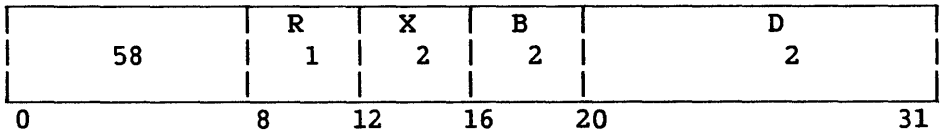
An overview of stack switching appears in Section 3.10. The format of a Linkage Table entry is described in Section 4.6.2. The format of Control Registers 6 and 7 is described in Section 4.6.3.

LOAD (LR, L)

LR R1,R2 (RR)



L R1,D2(X2,B2) (RX)



The second operand is placed in the first operand location, and the second operand is not changed. For the L instruction, the second operand must be fullword aligned.

Resulting Condition Code

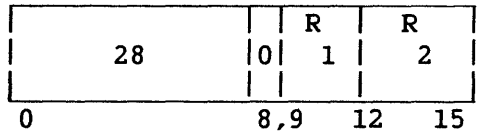
The condition code remains unchanged.

Program Exceptions

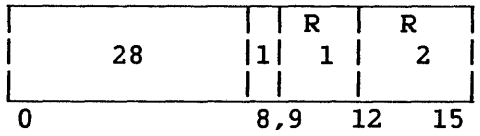
Access (fetch, operand 2 of L only)
Specification (L only)

LOAD (FLOATING-POINT) (LDR, LER, LD, LE)

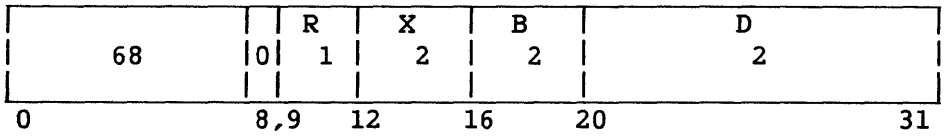
LDR R1,R2 (RR, Long)



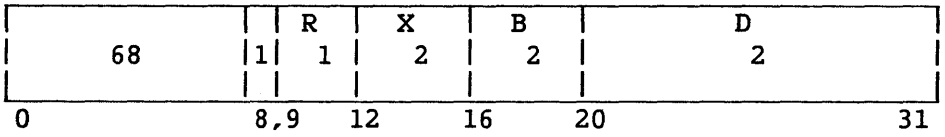
LER R1,R2 (RR, Short)



LD R1,D2(X2,B2) (RX, Long)



LE R1,D2(X2,B2) (RX, Short)



The second operand is placed in the first operand location and is not changed. Exponent overflow, exponent underflow, or lost significance cannot occur. For the LD instruction, the second operand must be fullword aligned.

Resulting Condition Code

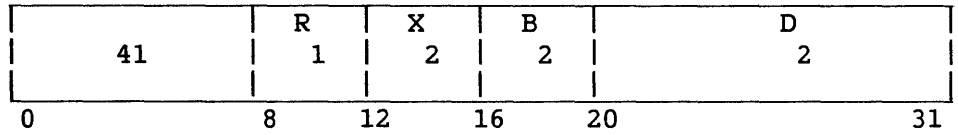
The condition code remains unchanged.

Program Exceptions

Addressing (LD, LE only)
Specification (LD, LE only)
Access (LD, LE only)

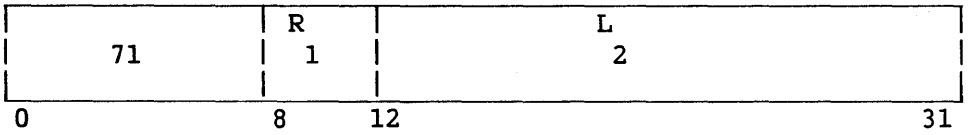
LOAD ADDRESS (LA)

LA R1,D2(X2,B2) (RX)



LOAD ADDRESS (RELATIVE) (RLA)

RLA R1,L2 (RL)



For LA, the address specified by the X2, B2, and D2 fields is inserted in bit positions 8-31 of the general register specified by the R1 field. For RLA, the address inserted is the sum of the current instruction address (bits 8-31 of the PCW) and the L2 field. Bits 0-7 of the register are set to 0s. The address computation follows the rules for base-displacement address formation. No memory references for operands take place, and the address is not inspected for access exceptions.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

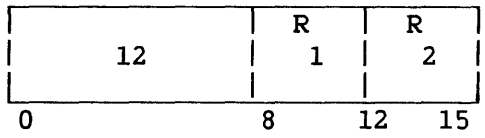
None

Programming Note

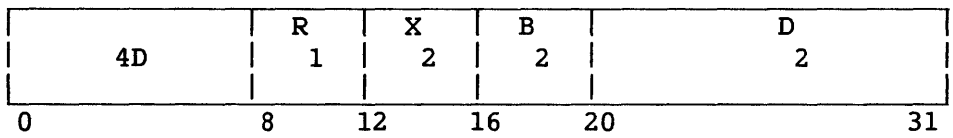
The same general register may be specified by the R1, X2, and B2 instruction field, except that General Register 0 can be specified only by the R1 field. In this manner it is possible to increment the low-order 24 bits of a general register other than General Register 0 by the contents of the D2 field of the instruction. The register to be incremented should be specified by R1 and by either X2 (with B2 set to 0) or B2 (with X2 set to 0).

LOAD AND TEST (LTR, LT)

LTR R1,R2 (RR)



LT R1,D2(X2,B2) (RX)



The second operand is placed in the first operand register, and its value determines the condition code. When the LT instruction is used, a fullword field from memory as specified by the second operand is loaded into the first operand register. The second operand is not changed.

The condition code of this instruction indicates whether the result is zero, or, if at least one bit of the result is on, whether the leftmost bit is on (called less than 0) or off (called greater than 0).

Operand 2 of the LT instruction requires fullword alignment.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is more than 0
3 --

Program Exceptions

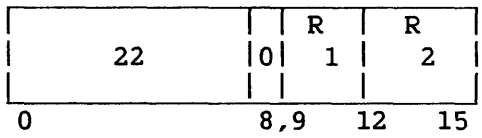
Access (fetch, operand 2, LT)
Specification (LT only)

Programming Note

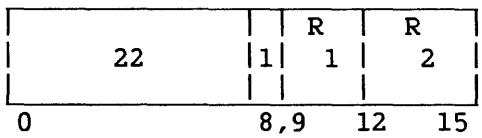
When the same register is specified as first and second operand location, the operation is equivalent to a test without data movement.

LOAD AND TEST (FLOATING-POINT) (LTDR, LTER)

LTDR R1,R2 (RR, Long)



LTER R1,R2 (RR, Short)



The second operand is placed in the first operand location, and its sign and magnitude determine the condition code. The second operand is not changed.

Resulting Condition Code

0 Result fraction is 0
1 Result is less than 0
2 Result is greater than 0
3 --

Program Exceptions

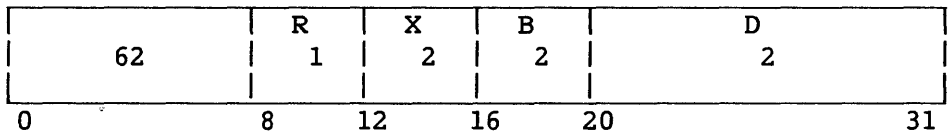
Specification

Programming Note

When the same register is specified as first and second operand location, the operation is equivalent to a test without data movement.

LOAD CHARACTER (LC)

```
LC   R1,D2(X2,B2)                (RX)
```



The second operand is placed in the first operand location. The second operand is one byte in length and is placed in the low-order byte of the first operand register. The three high-order bytes of the first operand register are set to binary 0s.

Resulting Condition Code

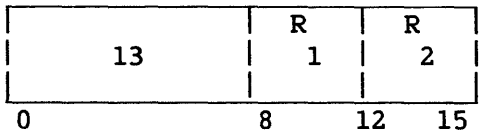
The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2)

LOAD COMPLEMENT (LCR)

LCR R1,R2 (RR)



The 2's complement of the second operand is placed in the first operand location.

The condition code of this instruction indicates whether the result is zero, or, if at least one bit of the result is on, whether the leftmost bit is on (called less than 0) or off (called greater than 0).

An overflow condition occurs when the maximum negative number is complemented. The number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is 1.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 Overflow

Program Exceptions

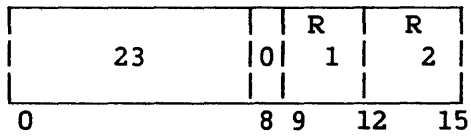
Fixed-point overflow

Programming Note

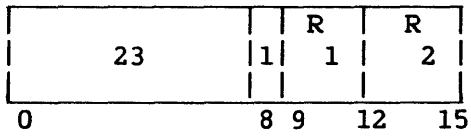
Zero and the maximum negative number do not have a 2's complement.

LOAD COMPLEMENT (FLOATING-POINT) (LCDR, LCER)

LCDR R1,R2 (RR, Long)



LCER R1,R2 (RR, Short)



The second operand is placed in the first operand location with the sign changed to the opposite value.

The sign bit of the second operand is inverted, while characteristic and fraction are not changed.

Resulting Condition Code

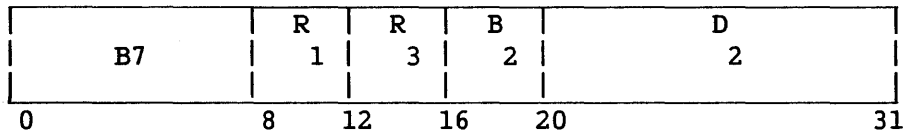
0 Result fraction is 0
1 Result is less than 0
2 Result is greater than 0
3 --

Program Exceptions

Specification

LOAD CONTROL (LCTL)[P]

```
LCTL   R1,R3,D2(B2)          (RS)
```



The set of control registers starting with the control register designated by the R1 field and ending with the control register designated by the R3 field is loaded from the locations designated by the second operand address.

The memory area from which the contents of the control registers are obtained starts at the location designated by the second operand address and continues through as many memory words as the number of control registers specified. The control registers are loaded in ascending order of their addresses, starting with the control register designated by the R1 field and continuing up to and including the control register designated by the R3 field. The second operand remains unchanged.

An attempt is made to fetch the operand from main memory for each of the designated control registers. Whenever the storage reference causes an access exception, the exception is indicated. The second operand must be designated on a word boundary; otherwise, a specification exception is recognized, and the operation is suppressed. A specification exception will also be recognized if R1 is numbered higher than R3 (wraparound).

Resulting Condition Code

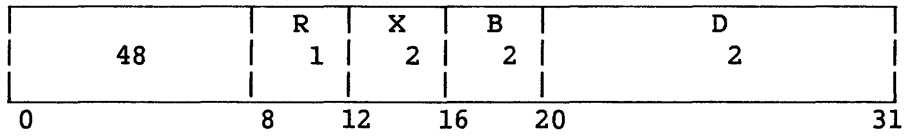
The condition code remains unchanged.

Program Exceptions

Privileged operation
Access (fetch, operand 2)
Specification

LOAD HALFWORD (LH)

```
LH   R1,D2(X2,B2)                (RX)
```



The second operand is placed in the first operand location, is two bytes in length, and is considered to be a 16-bit signed integer. It requires halfword alignment.

The second operand is expanded to 32 bits by propagating the sign-bit value to the 16 high-order bit positions. Expansion occurs after the operand is obtained from memory and before insertion in the register.

Resulting Condition Code

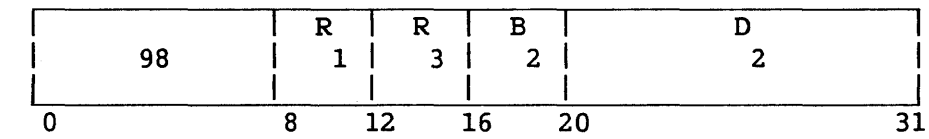
The condition code remains unchanged.

Program Exceptions

Specification
Access (fetch, operand 2)

LOAD MULTIPLE (LM)

LM R1,R3,D2(B2) (RS)



The set of general registers starting with the register specified by R1 and ending with the register specified by R3 is loaded from the locations designated by the second operand address.

The memory area from which the contents of the general registers are obtained starts at the location designated by the second operand address and continues through as many words as needed. The general registers are loaded in ascending order of their addresses, starting with the register specified by R1 and continuing up to and including the register specified by R3, with Register 0 following Register 15.

The second operand, which must be fullword aligned, remains unchanged.

Resulting Condition Code

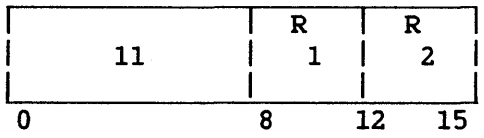
The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2)
Specification

LOAD NEGATIVE (LNR)

LNR R1,R2 (RR)



The 2's complement of the absolute value of the second operand is placed in the first operand location. The operation complements positive numbers; negative numbers remain unchanged. The number 0 remains unchanged with positive sign.

Resulting Condition Code

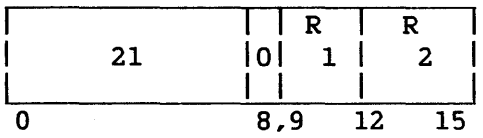
0 Result is 0
1 Result is less than 0
2 --
3 --

Program Exceptions

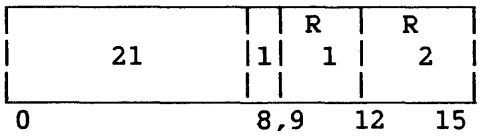
None

LOAD NEGATIVE (FLOATING-POINT) (LNDR, LNER)

LNDR R1,R2 (RR, Long)



LNER R1,R2 (RR, Short)



The second operand is placed in the first operand location with the sign made minus.

The sign bit of the second operand is made 1, even if the fraction is 0. Characteristic and fraction are not changed.

Resulting Condition Code

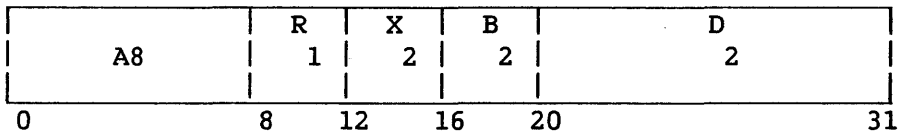
0 Result fraction is 0
1 Result is less than 0
2 --
3 --

Program Exceptions

Specification

LOAD OR TRAP (LOT)

```
LOT   R1,D2(X2,B2)              (RX)
```



The fullword field from memory as specified by the second operand is loaded into the first operand register.

The high-order bit of the word loaded is inspected. If this bit is 1, then a 'LOT' program interrupt is taken.

Resulting Condition Code

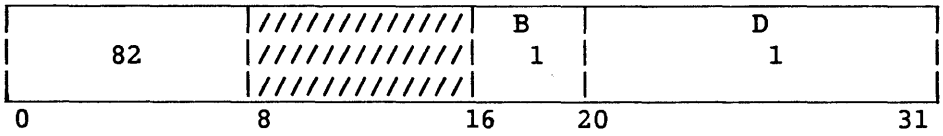
The condition code remains unchanged.

Program Exceptions

Specification
Access (fetch, operand 2)
'LOT' exception

LOAD PCW (LPCW)[P]

LPCW D1(B1) (S)



The two words at the location designated by the operand address replace the PCW. On the VS300, the operand must be halfword-aligned; on other VS systems, the operand must be word-aligned. Improper alignment of the operand causes the instruction to be suppressed with a specification exception.

The doubleword that is loaded becomes the PCW for the next sequence of instructions. This loads a new instruction address.

The interruption code of the new PCW is not retained as the PCW is loaded. When the PCW is subsequently stored because of an interruption, these bit positions contain a new code.

Resulting Condition Code

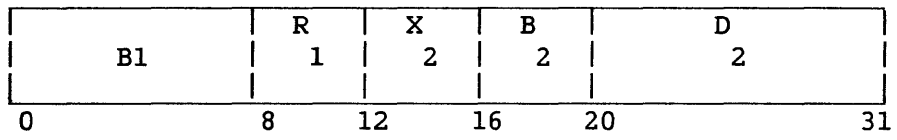
The condition code is set according to the condition code bits of the new PCW.

Program Exceptions

Specification
Privileged operation
Access (fetch, operand 1)

LOAD PHYSICAL ADDRESS (LPA)

```
LPA     R1,D2(X2,B2)            (RX)
```



The physical address corresponding to the second operand address is inserted in the general register designated by the R1 field. The remaining high-order bits of the register are set to 0.

The logical address specified by the X2, B2, and D2 fields is translated using the contents of the main memory page tables, which are located through the segment control registers and region tables. These translation structures are described in Section 4.3. The resultant 24-bit physical address is inserted in bit positions 8-31 of the general register designated by the R1 field, and Bits 0-7 are set to 0. The translated address is not inspected for protection or validity.

The condition code is set to 0 when translation can be completed; if the translation is not successfully completed, the condition code is set to a value of 1-3 and the general register designated by R1 is set to zero. The Page Fault Reporting Area (X'72') and Region Node Address area (X'74') in low memory are never modified by execution of LPA.

Resulting Condition Code

0 Successful translation

1 SCR specification or invalid virtual address
- Region Table address = 0
- Region Table address not word-aligned
- Page Table address (in Region Node) = 0
- Page Table address not word-aligned
- Virtual address not in any region
- Virtual address not in LOHI range of any Region Node

2 Page fault condition (fault bit of Page Table entry = 1)

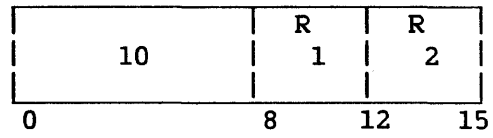
3 Page table fault condition
- Virtual address of Page Table faulted or invalid
- SCR recursion error case

Program Exceptions

None

LOAD POSITIVE (LPR)

```
LPR   R1,R2                          (RR)
```



The absolute value of the second operand is placed in the first operand location. The operation includes complementing of negative numbers; positive numbers remain unchanged.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The overflow causes a program interruption when the fixed-point overflow mask bit is 1.

Resulting Condition Code

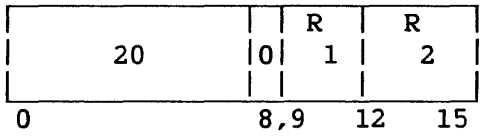
0 Result is 0
1 --
2 Result is greater than 0
3 Overflow

Program Exceptions

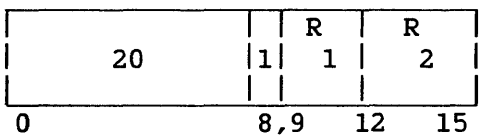
Fixed-point overflow
Specification

LOAD POSITIVE (FLOATING-POINT) (LPDR, LPER)

LPDR R1,R2 (RR, Long)



LPER R1,R2 (RR, Short)



The second operand is placed in the first operand location with the sign made positive.

The sign bit of the second operand is made 0, while the characteristic and fraction are not changed.

Resulting Condition Code

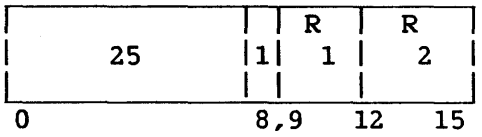
0 Result fraction is 0
1 --
2 Result is greater than 0
3 --

Program Exceptions

Specification

LOAD ROUNDED (FLOATING-POINT) (LRER)

LRER R1,R2 (RR, Short)



The second operand is rounded to short format, and the result is placed in the first operand location.

Rounding consists of adding 1 to bit position 32 of the long second operand, and propagating the carry, if any, to the left. The sign of the fraction is ignored, and addition is performed as if the fraction were positive.

If rounding causes a carry out of the high-order digit position of the fraction, the fraction is shifted right by one digit position, and the characteristic is increased by 1.

The sign of the result is the same as the sign of the second operand. No normalization takes place.

An exponent overflow exception is recognized when shifting the fraction right causes the characteristic to exceed 127. The operation is completed by loading a number whose characteristic is 128 less than the correct value, and a program interruption for exponent overflow occurs. The result is normalized, and the sign and fraction remain correct.

Exponent underflow and significance exceptions cannot occur.

Resulting Condition Code

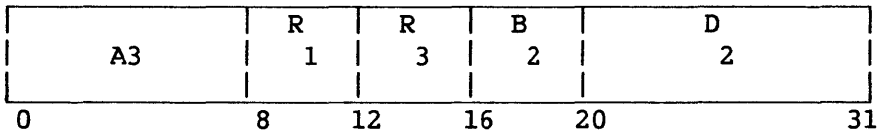
The condition code remains unchanged.

Program Exceptions

Exponent overflow
Specification

LOAD SEGMENT CONTROL REGISTER (LSCTL)[P]

LSCTL R1,R3,D2(B2) (RS)



The set of segment control registers (SCRs) starting with the register specified by R1 and ending with the register specified by R3 is loaded from locations designated by the second operand address. Allowable SCR values are 0,2,4, and 6.

The memory area from which the contents of the SCRs are obtained starts at the location designated by the second operand address and continues through as many words as needed. Each SCR requires eight bytes of data from memory.

The SCRs are loaded in ascending order of their addresses, starting with the register specified by R1 and continuing up to and including the register specified by R3. R3 must be greater than or equal to R1. The contents of the memory area remain unchanged.

Operand 2 requires fullword alignment.

Resulting Condition Code

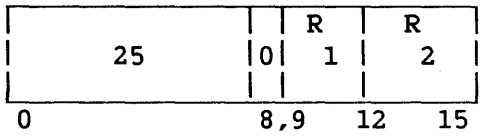
The condition code remains unchanged.

Program Exceptions

Access (store, operand 2)
Privileged operation
Specification

LOAD SHORT TO LONG (FLOATING-POINT) (LDER)

LDER R1,R2 (RR, Long)



The second operand is extended with low-order 0s to long format, and the result is placed in the first operand location.

Resulting Condition Code

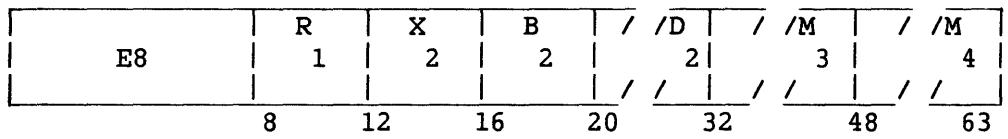
The condition code remains unchanged.

Program Exceptions

None

MODIFY COUNTER (MCOUNT)

MCOUNT R1,D2(X2,B2),M3,M4 (Special)



R1 is a general register into which the main memory word addressed by the second operand is copied after being modified by addition or subtraction. The second operand addresses a word in main memory, which must be fullword-aligned. M3 is a 16 bit function mask that determines the operation performed by MCOUNT. M4 is a 16-bit unsigned integer used as an increment for the add immediate and subtract immediate functions.

The following functions are available using the M3 mask values given below:

X'8000' Add 1 to main memory word; result to R1.

X'4000' Subtract 1 from main memory word; result to R1.

X'2000' Exchange R1 with main memory word; unmodified main memory word to R1.

X'1000' Add M4 to main memory word; result to R1

X'0800' Subtract M4 from main memory word; result to R1

Invalid values for the function mask result in a specification exception. In all cases except the exchange function (X'2000'), the condition code reports on the new value of the main memory word; and the general register named by R1 is loaded with this new value if the named register is not General Register 0. In the case of the exchange function, the condition code reports on the updated value of the general register named by R1; and that register is always updated, even if it is Register 0.

Resulting Condition Code

0 Result = 0
1 Result is negative
2 Result is positive

Program Exceptions

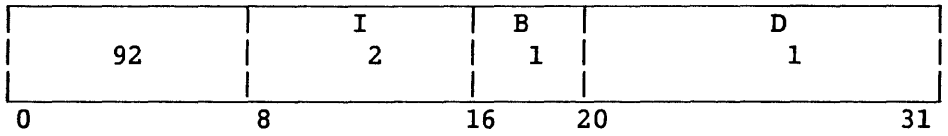
Specification
Overflow

Programming Note

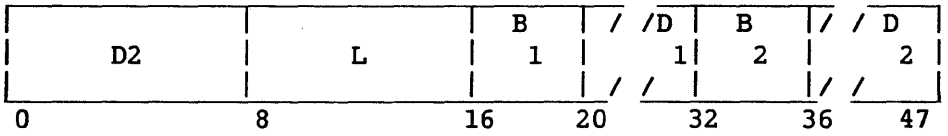
An overflow can occur when adding or subtracting, and will cause a program check, whatever the setting of the binary arithmetic overflow mask. On overflow, the main memory word is not modified, and the condition code is undefined. Given a 32-bit counter (32 bits in 2's complement =+/- 10**9), normal usage of this instruction should never cause overflow.

MOVE (MVI, MVC)

MVI D1(B1),I2 (SI)



MVC D1(L,B1),D2(B2) (SS)



The second operand is placed in the first operand location.

The SS format is used for a storage-to-storage move. In the MVC instruction, the length field in the instruction format is the operand length minus 1. The SI format introduces one 8-bit byte from the instruction stream.

In storage-to-storage movement the fields may overlap in any desired way. Movement is left to right through each field a byte at a time.

The bytes to be moved are not changed or inspected.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

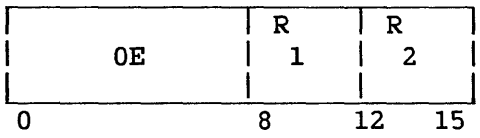
Access (fetch, operand 2 of MVC; store operand 1, MVI and MVC)

Programming Note

It is possible to propagate one character through an entire field by having the first operand field start one character to the right of the second operand field.

MOVE CHARACTERS LONG (MVCL)

MVCL R1,R2 (RR)



The second operand is placed in the first operand location, provided overlapping of operand locations does not affect the final contents of the first operand location.

The R1 and R2 fields each designate an even-odd pair of general registers and must each specify an even-numbered register; otherwise, a specification exception is recognized.

The leftmost bytes of the first operand and second operand locations are designated by the contents of bit positions 8-31 of the general registers specified by the R1 and R2 fields, respectively. The numbers of bytes in the first operand and second operand locations are specified by the contents of bit positions 8-31 of general registers having addresses R1+1 and R2+1, respectively. Bit positions 0-7 of register R2+1 contain the padding character. The contents of bit positions 0-7 of registers R1, R1+1, and R2 are ignored.

The movement starts at the high-order end of both fields and proceeds to the right. The bytes to be moved are not changed or inspected. The operation is ended when the number of bytes specified by bit positions 8-31 of register R1+1 have been moved into the first operand location. If the second operand is shorter than the first operand, the remaining low-order bytes of the first operand are filled with the padding character.

As part of the execution of the instruction, the values of the two count fields are compared for the setting of the condition code, and a check is made for destructive overlap of the operands. Operands are said to overlap destructively when the first operand location is used as a source after data has been moved into it, assuming movement to be performed one byte at a time. The inspection for overlap is performed by use of logical operand addresses. When the operands overlap destructively, no movement takes place and condition code 3 is set. Movement is performed when the high-order byte of the first operand coincides with or is to the left of the high-order byte of the second operand, or if the high-order byte of the first operand is to the right of the rightmost second operand byte participating in the operation. The rightmost second operand byte is determined by using the smaller of the first operand and second operand counts.

When the count specified by bit positions 8-31 of register R1+1 is 0, no movement takes place, and the condition code is set to 0 or 1 to indicate the relative values of the counts.

The execution of the instruction is interruptible. When an interruption occurs after a unit of operation other than the last one, the contents of registers R1+1 and R2+1 are decremented by the number of bytes moved and the contents of registers R1 and R2 are incremented by the same number, so that the instruction, when re-executed, resumes at the point of interruption. The high-order bytes of registers R1 and R2 are set to 0; the contents of the high-order byte of registers R1+1 and R2+1 remain unchanged. If the operation is interrupted during padding, the count field in register R2+1 is 0, the address in register R2 is incremented by the original contents of register R2+1, and the contents of registers R1 and R1+1 reflect the extent of the padding operation.

The instruction may be refetched from main storage even in the absence of an interruption during execution.

At the completion of the operation, the count in register R1+1 is 0 and the address in register R1 is incremented by the original value of the count in register R1+1. The count in register R2+1 is decremented by the number of bytes moved out of the second operand location, and the address in register R2 is incremented by the same amount. The contents of bit positions 0-7 of registers R1 and R2 are set to 0, even in the case when one or both of the original count values are 0 or when condition code 3 is set. The contents of bit positions 0-7 of registers R1+1 and R2+1 remain unchanged.

When the count specified by bit positions 8-31 of register R1+1 is 0, or condition code 3 is set, no exceptions associated with operand access are recognized. When the count specified by bit positions 8-31 of register R2+1 for the second operand is larger than that for the first operand, access exceptions are not recognized for the part of the second operand field that is in excess of the first operand field.

Resulting Condition Code

0 First operand and second operand counts are equal
1 First operand count is low
2 First operand count is high
3 No movement performed because of destructive overlap

Program Interruptions

Access (fetch, operand 2; store, operand 1)
Specification

Programming Notes

When the first operand count is 0, the operation consists of setting the condition code and setting the high-order bytes of registers R1 and R2 to 0.

When the contents of the R1 and R2 fields are identical, the condition code is set to 0, but protection and addressing exceptions are not indicated when called for by the operand designation.

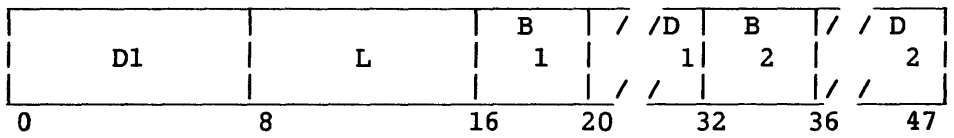
Since the execution of MOVE LONG is interruptible, the instruction cannot be used for situations where the program must rely on uninterrupted execution of the instruction or on the clock's not being updated during the execution of the instruction. Similarly, the program should normally not let the first operand of MOVE LONG include the location of the instruction. This is because the new contents of the location may be interpreted as the instruction if execution is resumed after an interruption or if the instruction is refetched without an interruption.

Special precautions should be taken when MOVE LONG is made the subject of an EXECUTE instruction. See the programming notes in the description of the EXECUTE instruction.

When the CONTROL MODE button is pressed during the execution of MOVE LONG or COMPARE LOGICAL LONG, the CP enters Control mode at the completion of the execution of the next unit of operation. If the modification trap condition occurs during the current unit of operation, the trap will be taken at the completion of execution of the current unit. However, the single-step trap will only be taken at the completion of the instruction; it will not be taken at the completion of any other unit of execution. The amount of data processed in a unit of operation may depend on the particular condition that caused the execution of the instruction to be interrupted.

MOVE NUMERICS (MVN)

```
MVN   D1(L,B1),D2(B2)          (SS)
```



The low-order four bits of each byte in the second operand field, the numerics, are placed in the low-order bit positions of the corresponding bytes in the first operand field.

L is the length of each operand, minus 1.

The instruction is storage-to-storage. Movement is from left to right through each field, one byte at a time, and the fields may overlap in any desired way.

The numerics are not changed or checked for validity. The high-order four bits of each byte, the zones, remain unchanged in both operand fields.

Resulting Condition Code

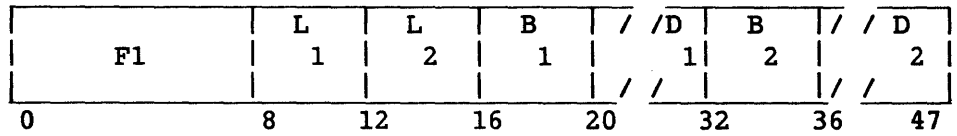
The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2; store, operand 1)

MOVE WITH OFFSET (MVO)

MVO D1(L1,B1),D2(L2,B2) (SS)



The second operand is placed to the left of and adjacent to the low-order four bits of the first operand.

The low-order four bits of the first operand are attached as low-order bits to the second operand; the second operand bits are offset by four bit positions, and the result is placed in the first operand location. The first operand and second operand bytes are not checked for valid codes.

The fields are processed right to left. If necessary, the second operand is extended with high-order 0s. If the first operand field is too short to contain all bytes of the second operand, the remaining information is ignored. Overlapping fields may occur and are processed by storing a result byte as soon as the necessary operand bytes are fetched.

L1 and L2 are the operand lengths, minus 1.

Resulting Condition Code

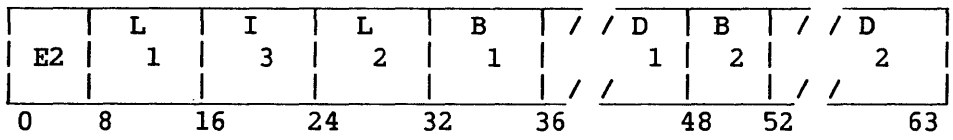
The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2; fetch and store, operand 1)

MOVE WITH PAD (MVPC)

MVPC D1(L1,B1),D2(L2,B2),I3 (SSI)



The second operand is placed in the first operand location. If the first operand length (L1) is less than the second operand length (L2), only the number of bytes specified by L1 is moved. If the first operand length is greater than the second operand length, the additional bytes of the first operand are filled with the character specified in the I3 field of the instruction.

The bytes to be moved are not changed or inspected.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

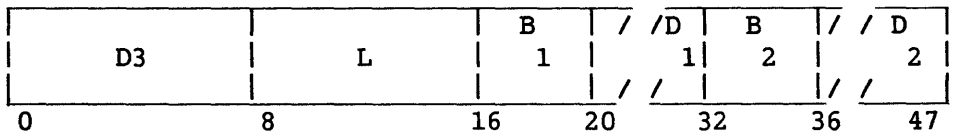
Access (fetch, operand 2; store, operand 1)

Programming Note

At least one byte of the first operand is always moved by a successful MOVE WITH PAD. (The L1 and L2 fields of the instruction are 1 less than the lengths they specify.)

MOVE ZONES (MVZ)

MVZ D1(L,B1),D2(B2) (SS)



The high-order four bits of each byte in the second operand field (the zones) are placed in the high-order four bit positions of the corresponding bytes in the first operand field.

The instruction is storage-to-storage. Movement is from left to right through each field one byte at a time, and the fields may overlap in any desired way.

The zones are not changed or checked for validity. The low-order four bits of each byte (the numerics) remain unchanged in both operand fields.

L is the length of each operand, minus 1.

Resulting Condition Code

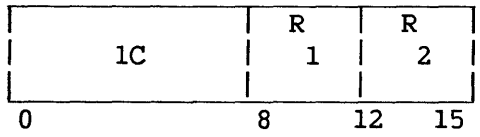
The condition code remains unchanged.

Program Exceptions

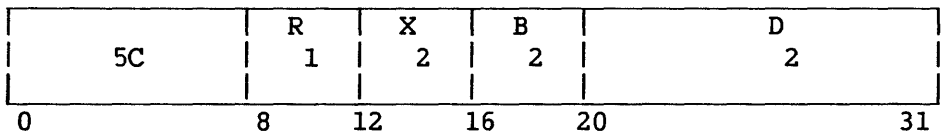
Access (fetch, operand 2; fetch and store, operand 1)

MULTIPLY (MR, M)

MR R1,R2 (RR)



M R1,D2(X2,B2) (RX)



The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand. For the M instruction, operand 2 requires fullword alignment.

Both multiplier and multiplicand are 32-bit signed integers. The product is always a 64-bit signed integer and occupies register R1 and the register following R1. The multiplicand is taken from the register following R1. The contents of register R1 (replaced by the high-order part of the product) are ignored. An overflow cannot occur.

The R1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R1 is odd.

The sign of the product is determined by the rules of algebra, except that a result of 0 is always positive.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

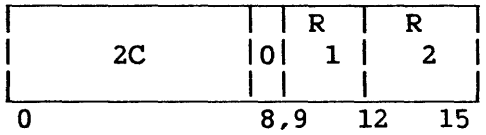
Specification
Access (fetch, operand 2 of M only)

Programming Note

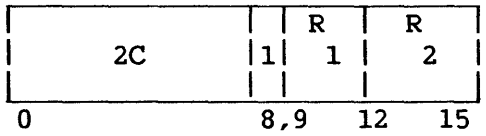
The significant part of the product usually occupies 62 or fewer bits. Only when two maximum negative numbers are multiplied are 63 significant product bits formed. Since 2's-complement notation is used, the sign bit is extended right until the first significant product digit is encountered.

MULTIPLY (FLOATING-POINT) (MDR, MER, MD, ME)

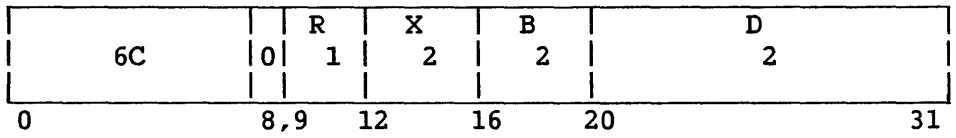
MDR R1,R2 (RR, Long)



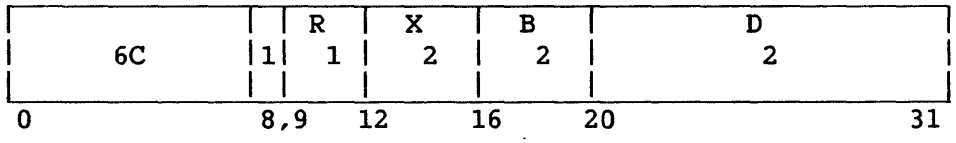
MER R1,R2 (RR, Short)



MD R1,D2(X2,B2) (RX, Long)



ME R1,D2(X2,B2) (RX, Short)



The normalized product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

The multiplication of two floating-point numbers consists of adding the characteristics and multiplying the fractions. The sum of the characteristics less 64 is used as the characteristic of an intermediate product. The sign of the product is determined by the rules of algebra.

The product fraction is normalized by prenormalizing the operands and postnormalizing the intermediate product when necessary. The intermediate sum of the characteristics is reduced by the number of left-shifts. The intermediate product of the fractions is truncated to 15 digits before the left-shifting.

Exponent overflow occurs if the final sum of the characteristics exceeds 127. The operation is completed and a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is smaller by 128 than the correct characteristic. The overflow exception does not occur for an intermediate sum of characteristics exceeding 127 when the final characteristic is brought within range because of normalization.

Exponent underflow occurs if the final sum of the characteristics is less than 0. If the corresponding mask bit is 1, a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is larger by 128 than the correct characteristic. If the corresponding mask bit is not 1, the result is made a true 0. Underflow is not signaled when an operand's characteristic becomes less than 0 during prenormalization, and the correct characteristic and fraction value are used in the multiplication.

When all 15 digits of the intermediate fraction are 0, the product, sign, and characteristic are all made 0, yielding a true zero result. No interruption for exponent underflow or exponent overflow can occur when the result fraction is 0. The program interruption for lost significance is never taken for multiplication.

The second operand of the MD and ME instructions requires fullword alignment.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

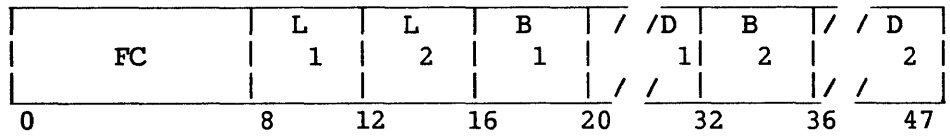
Specification
Exponent overflow
Exponent underflow
Access (MD and ME only)

Programming Note

Interchanging the two operands in a floating-point multiplication does not affect the value of the product.

MULTIPLY DECIMAL (MP)

MP D1(L1,B1),D2(L2,B2) (SS)



The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

The multiplier size is limited to 15 digits plus a sign and must be less than the multiplicand size. L1 and L2 are the lengths of the operands, minus 1. Length code L2, when larger than 7 or larger than or equal to the length code L1, causes a specification exception; in this case, the operation is suppressed and a program interruption occurs.

Since the number of digits in the product is the sum of the number of digits in the operands, the multiplicand must have high-order zero digits for a field size at least equal to the multiplier field size; otherwise, a data exception is recognized and a program interruption occurs. This definition of the multiplicand field insures that no product overflow can occur. The maximum product size is 31 digits. At least one high-order digit of the product field must be 0.

All operands and results are treated as signed integers, right-aligned in their field. The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs, even if one or both operands are 0.

The multiplier and product fields may overlap when their least significant bytes coincide.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

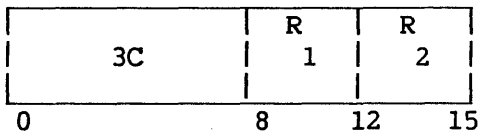
Access (fetch, operand 2; store, operand 1)
Data
Specification

Programming Note

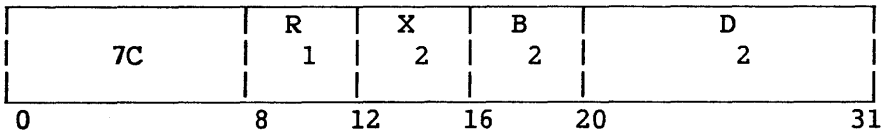
When the multiplicand does not have the desired number of most significant 0s, multiplication may be preceded by a ZERO AND ADD into a larger field.

MULTIPLY DECIMAL (FLOATING-POINT) (MQR, MQ)

MQR R1,R2 (RR)



MQ R1,D2(X2,B2) (RX)



The normalized product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand. Fullword alignment is required.

The multiplication of two decimal floating-point numbers consists of a characteristic addition and a fraction multiplication. The sum of the characteristics minus 64 is used as the characteristic of an intermediate product. The sign of the product is determined by the rules of algebra.

The product fraction is normalized by prenormalizing the operands and postnormalizing the intermediate product, if necessary. Postnormalizing is performed after the fraction is truncated to 15 digits.

Exponent overflow occurs if the final product characteristic exceeds 127. The operation is completed and a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is 128 smaller than the correct characteristic. The overflow exception does not occur for an intermediate product characteristic exceeding 127 when the final characteristic is brought within range because of postnormalization.

Exponent underflow occurs if the final product characteristic is less than zero. If the corresponding mask bit is 1, a program interruption occurs. The fraction is normalized and correct, the sign is correct, and the characteristic is 128 larger than the correct characteristic. If the corresponding mask bit is 0, the result is made a true zero. Underflow is not signaled when an operand's characteristic becomes less than zero during prenormalization, and the correct characteristic and fraction value are used in the multiplication.

When all 15 digits of the intermediate product fraction are 0s, the product is made a true zero. No interruption for exponent underflow or exponent overflow can occur when the result fraction is zero. The program interruption for lost significance is never taken for multiplication.

Resulting Condition Code

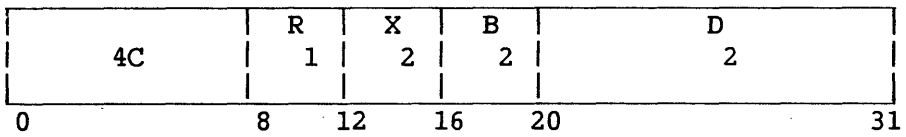
The condition code remains unchanged.

Program Exceptions

Specification
Data
Exponent overflow
Exponent underflow
Access (MQ only)

MULTIPLY HALFWORD (MH)

MH R1,D2(X2,B2) (RX)



The product of the second operand (multiplier) and first operand (multiplicand) replaces the multiplicand. The second operand is two bytes in length, must be halfword aligned, and is considered to be a 16-bit signed integer.

Both multiplicand and product are 32-bit signed integers and may be located in any general register. The 16-bit multiplier is expanded to 32 bits before multiplication by propagating the sign-bit value through the 16 high-order bit positions. The multiplicand is replaced by the low-order part of the product. The bits to the left of the 32 low-order bits are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs, except that a result of 0 is always positive.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

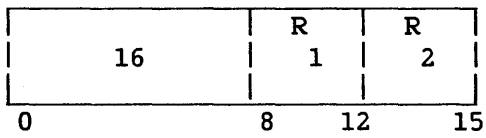
Access (fetch, operand 2)
Specification

Programming Note

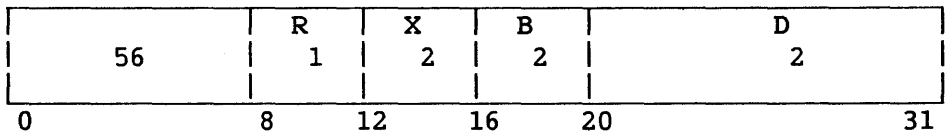
The significant part of the product usually occupies 46 or fewer bits; however, 47 bits are occupied when both operands have the maximum negative value. Since the low-order 32 bits of the product are stored unchanged, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product if there is overflow.

OR (OR, O, OI, OC)

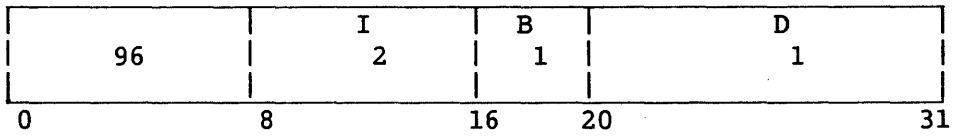
OR R1,R2 (RR)



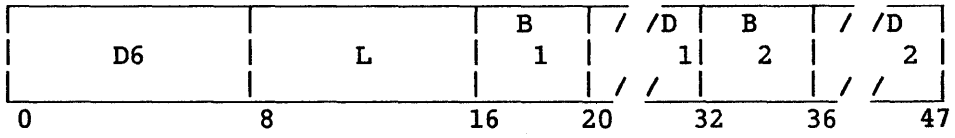
O R1,D2(X2,B2) (RX)



OI D1(B1),I2 (SI)



OC D1(L,B1),D2(B2) (SS)



The logical sum (OR) of the bits of the first and second operands is placed in the first operand location. Operands are treated as unstructured logical quantities, and the connective inclusive OR is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit position in one or both operands contains a 1; otherwise, the result bit is set to 0. All operands and results are valid. Operand 2 of the O instruction requires fullword alignment.

Resulting Condition Code

0 Result is 0
1 Result not 0
2 --
3 --

Program Exceptions

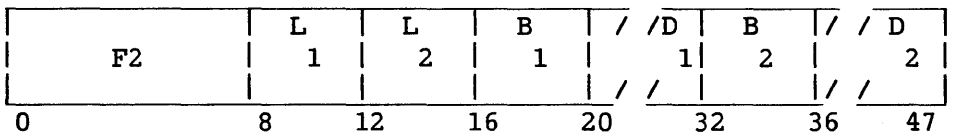
Specification (O only)

Access (fetch, operand 2, O and OC; fetch and store, operand 1, OI and OC)

Programming Note

The OR may be used to set a bit to 1. For this purpose, the second operand should have 1s in all positions corresponding to the first operand bits to be set to 1.

PACK D1(L1,B1),D2(L2,B2) (SS)



The format of the second operand is changed from zoned to packed, and the result is placed in the first operand location.

The second operand is assumed to have the zoned format. All zones are ignored except the zone over the low-order digit, which is assumed to represent a sign. The sign is placed in the rightmost four bits of the low-order byte, and the digits are placed adjacent to the sign and to each other in the remainder of the result field. The sign and digits are moved unchanged to the first operand field, and are not checked for valid codes.

The fields are processed right to left. If necessary, the second operand is extended with high-order 0s. If the first operand field is too short to contain all significant digits of the second operand field, the remaining high-order digits are ignored. Overlapping fields may occur and are processed by storing one result byte immediately after the necessary second operand bytes are fetched. Except for the rightmost byte of the result field, which is stored immediately upon fetching the rightmost byte of the second operand, two operand bytes are needed for each result byte.

L1 and L2 are the operand lengths, minus 1.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

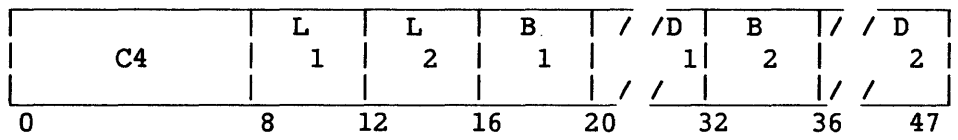
Access (fetch, operand 2; store, operand 1)

Programming Notes

The PACK instruction may be used to interchange the two digits in one byte by specifying a 0 in the L1 and L2 fields and the same address for both operands.

To remove the zones of all bytes of a field, including the low-order byte, both operands must be extended with a dummy byte in the low-order position, which subsequently is ignored in the result field.

PAL D1(L1,B1),D2(L2,B2) (SS)



The format of the second operand is changed from external format to packed, and the result is placed in the first operand location.

The second operand is assumed to have the external format. The source field may contain (moving right to left) blanks (ASCII space code), followed by a sign (ASCII plus or minus), followed by data. Or it may contain data followed by a sign followed by blanks. It may also contain a single ASCII decimal point character. L1 and L2 are the operand lengths, minus 1.

The source field is scanned right to left until the first nonblank character is encountered. If the first nonblank character is a valid sign character (hexadecimal 2B for plus or hexadecimal 2D for minus) its packed equivalent (1111 for plus or 1101 for minus) will be stored in the rightmost four bits of the least significant byte of the first operand (receiver field). If a decimal point has not previously been encountered, a check for decimal point is made. If the character is a decimal point, its existence and position will be reflected in the contents of Register 1, and the recognition of another decimal point will be treated as an invalid character. A final test is made to determine whether or not the character is a valid decimal character. The scan continues until the leftmost source byte is reached. If the first nonblank character was not a valid sign, the leftmost character is checked for a valid sign, and the first operand is set accordingly. If no sign is specified in the source field, a plus sign (1111) is stored in the first operand.

Table 8-3 summarizes the scan order, disregarding any leading or trailing blanks.

Table 8-3. PACK AND ALIGN Scan Order

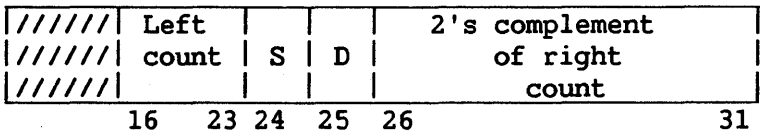
Order	Test	When Applied
1	Sign	To rightmost nonblank character
2	Sign	To leftmost character (if the rightmost nonblank character was not a sign)
3	Decimal point	Until a decimal point is encountered
4	Valid decimal character	Always

The code coversions from external ASCII to packed format are as follows:

- ASCII codes 0 through 9 are converted to 4-bit binary equivalents.
- The sign character, if present, is converted to 1111 for plus and 1101 for minus, and is stored in the rightmost four bits of the packed field. The presence of the sign character in the source field is indicated in Register 1. If no sign was found, a plus code is stored in the sign position of the packed field.
- The decimal point presence and position is indicated in Register 1 and it is skipped.
- The destination field is padded with leading packed 0s.
- The number of source digits converted is indicated in Register 1. Leading 0s preceding the decimal point in the source field are counted as digits.

The target field is filled in from right to left. If necessary, the second operand is padded with most significant 0s. If the first operand is too short to contain all significant digits of the second operand field, the remaining digits are ignored and the condition code is set to indicate truncation. In all cases of truncation, the operation is completed ignoring truncated digits. If an invalid character is encountered during conversion, the condition code is set to indicate a data validity error and the instruction is terminated.

If the instruction is completed, Register 1 is set to reflect the result of the operation. Bit 24 of Register 1 is set if a sign character was present, and Bit 25 is set if a decimal point was present. Bits 0 through 15 of Register 1 are unchanged. Bits 16 through 23 are set to the number of digits (including 0s) to the left of the decimal point in the source field. Bits 26-31 are set to the 2's complement of the count of digits (including 0s) to the right of the decimal point in the source field. Register 1 appears as follows after the PAL instruction is completed without encountering an invalid character:



R1 Bits	Function
16-23	Left count: count of source digits (including 0s) to left of decimal point
24	S, Sign presence 0 = no sign present 1 = sign present
25	D, Decimal point presence 0 = no source field decimal point encountered 1 = decimal point encountered
26-31	2's complement of right count: 2's complement of count of digits to right of the effective decimal point in the source field

Overlapping operand fields will yield unpredictable results.

Resulting Condition Code

0 Conversion completed successfully
1 Invalid character encountered
2 --
3 Left truncation occurred

Programming Exceptions

Access (fetch, operand 2; store, operand 1)

Programming Notes

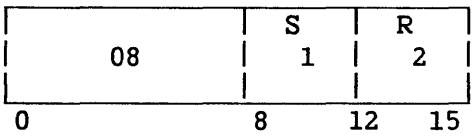
The initial contents of Register 1 are overwritten by the action of PAL.

If all digits in the source field are 0, the sign of the result will reflect the sign of the source. If truncation of a nonzero field occurs, the sign will reflect the value before truncation.

The "right count" in Register 1 after execution of PAL is in 2's-complement form for use in the second operand of the SHIFT AND ROUND DECIMAL (SRP) instruction.

POP (POP)

POP S1,R2 (RR)



The relevant stack vector is determined from the S1 field of the instruction. The stack pointer is incremented by 4. Register R2 is then loaded with the contents of the four bytes which were addressed by the stack pointer before updating.

Resulting Condition Code

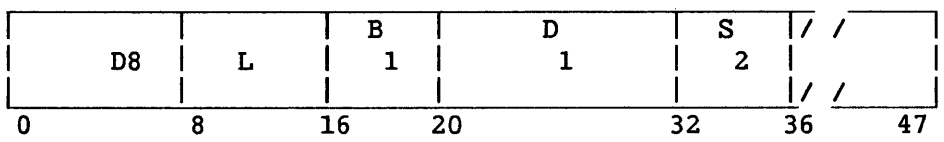
The condition code remains unchanged.

Program Exceptions

Specification
Access (fetch, bytes popped from stack)

POP CHARACTERS (POPC)

POPC D1(L,B1),0(S2) (SS)



The relevant stack vector is determined from the S2 field of the instruction. The D2 field is ignored. Bytes are taken from the location addressed by the stack pointer and ascending locations, and stored in ascending locations beginning at the location specified by the B1 and D1 fields. The number of bytes specified is stored. The stack pointer is then incremented by this number. The stack pointer is then incremented again (by 0, 1, 2, or 3) so that it addresses a fullword boundary.

L is the operand length in bytes, minus 1.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

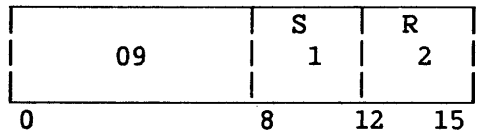
Specification
Access (fetch, bytes popped from stack; store, operand 2)

Programming Note

This is a move from a word-aligned location to a location with no alignment restriction.

POP HALFWORD (POPH)

POPH S1,R2 (RR)



The relevant stack vector is determined from the S1 field of the instruction. The stack pointer is incremented by 4. Then the low-order halfword of register R2 is loaded with the contents of the two low-order bytes of the word that was addressed by the stack pointer before updating. Bit 16 of register R2 is then propagated through the high-order half (Bits 0 through 15) of the register.

Resulting Condition Code

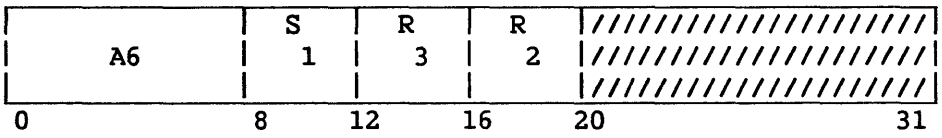
The condition code remains unchanged.

Program Exceptions

Specification
Access (fetch, bytes popped from stack)

POP MULTIPLE (POPM)

POPM S1,R3,R2 (RS)



The relevant stack vector is determined from the S1 field of the instruction. The stack pointer is incremented by the number of bytes implied by the range of registers R3 to R2. Register R3 and succeeding registers (with Register 0 following Register 15) are then loaded, starting from the location that was addressed by the stack pointer before updating, until register R2 has been loaded.

Resulting Condition Code

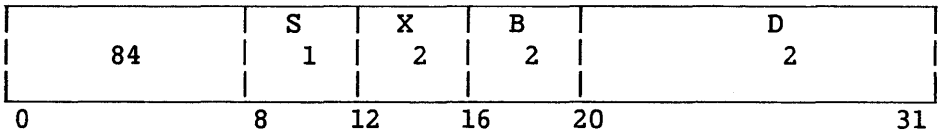
The condition code remains unchanged.

Program Exceptions

Specification
Access (fetch, bytes popped from stack)

POP NOTHING (POPN)

POPN S1,D2(X2,B2) (RX)



The relevant stack vector is determined from the S1 field of the instruction. The D2(X2,B2) value is added to the address in the stack pointer and the result stored in the stack pointer. The stack pointer is then incremented (by 0, 1, 2, or 3) so that it addresses a fullword boundary.

Resulting Condition Code

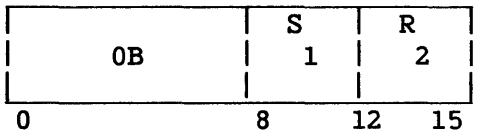
The condition code remains unchanged.

Program Exceptions

Specification

PUSH (PUSH)

PUSH S1,R2 (RR)



The relevant stack vector is determined from the S1 field of the instruction. The contents of the register specified by the R2 field are stored at the location addressed by the stack pointer, minus 4. The stack pointer of the stack vector is then decremented by 4.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

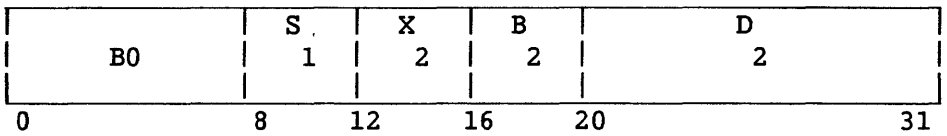
Stack overflow
Specification
Access (store, bytes pushed onto stack)

Programming Note

If the value in the stack pointer is pushed onto a stack by PUSH or PUSHM, the value pushed will be that in the register before the instruction was executed.

PUSH ADDRESS (PUSHA)

PUSHA S1,D2(X2,B2) (RX)



The relevant stack vector is determined from the S1 field of the instruction. The address in the stack pointer is decremented by 4. The second operand address is then placed in the three low-order bytes of the word addressed by the stack pointer. The high-order byte of this word is set to binary 0s. Address computation follows the rules for base-displacement address formation.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

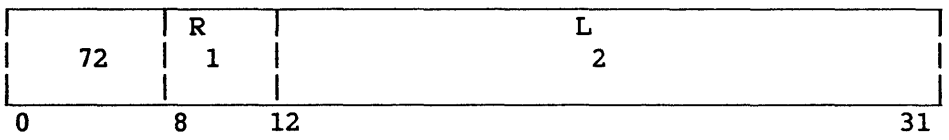
Stack overflow
Specification
Access (store, bytes pushed onto stack)

Programming Note

The second operand address is determined before the stack pointer is decremented, and therefore reflects the contents of registers before the instruction is executed.

PUSH ADDRESS (RELATIVE) (RPUSHA)

RPUSHA R1,L2 (RL)



The sign of the L2 field is extended 12 bits to the left, to form a 32-bit signed 2's-complement displacement. The displacement is added to the current instruction address to form the effective address.

Instruction execution is then identical to the corresponding RX instruction.

When the instruction is executed, the current instruction address used in the effective-address calculation is the address of the EXECUTE instruction.

Resulting Condition Code

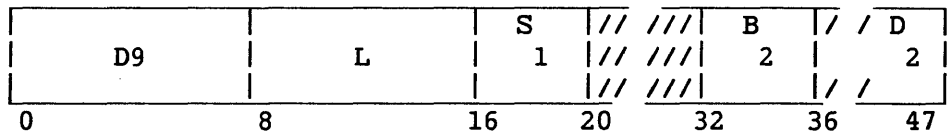
The condition code remains unchanged.

Program Exceptions

Stack overflow
Specification
Access (store, bytes pushed onto stack)

PUSH CHARACTERS (PUSHC)

PUSHC 0(L,S1),D2(B2) (SS)



The relevant stack vector is determined from the S1 field of the instruction. The length specified is subtracted from the stack pointer in the stack vector. The stack pointer is then decremented again (by 0, 1, 2, or 3) until it addresses a fullword boundary. Bytes are then taken from the location specified by the B2 and D2 fields and ascending locations; they are stored in ascending locations beginning at the location addressed by the updated stack pointer. The number of bytes specified is stored.

L is the operand length, minus 1.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

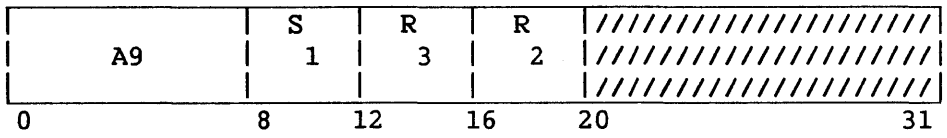
Stack overflow
Specification
Access (store, bytes pushed onto stack; fetch, operand 2)

Programming Note

This is a move from a location with no alignment restriction to a word-aligned location.

PUSH MULTIPLE (PUSHM)

```
PUSHM   S1,R3,R2          (RS)
```



The relevant stack vector is determined from the S1 field of the instruction. The values in register R2 and preceding registers (with Register 15 preceding Register 0) are stored in descending words starting four bytes below the location addressed by the stack pointer, until the register specified by the R3 field has been stored. The stack pointer is then decremented by the number of bytes stored.

Resulting Condition Code

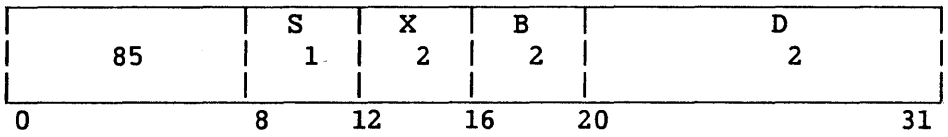
The condition code remains unchanged.

Program Exceptions

Stack overflow
Specification
Access (store, bytes pushed onto stack)

PUSH NOTHING (PUSHN)

PUSHN S1,D2(X2,B2) (RX)



The relevant stack vector is determined from the S1 field of the instruction. The D2(X2,B2) value is subtracted from the address in the stack pointer and the result is stored in the stack top word. The stack pointer is then decremented again (by 0, 1, 2, or 3) until it addresses a fullword boundary.

Resulting Condition Code

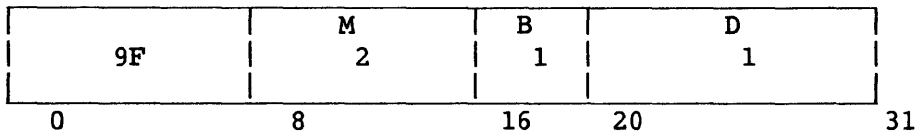
The condition code remains unchanged.

Program Exceptions

Stack overflow
Specification

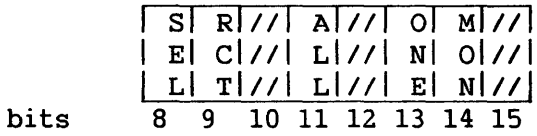
RESET REFERENCE AND CHANGE BITS (RRCB)[P]

```
RRCB   D1(B1),M2              (SI)
```



This privileged instruction, according to the value of M2, either examines and modifies the Reference and Change Table (RCT), or modifies the T-RAM and Monitor Area. When used with the RCT, operand 1 must be a physical address; when used with the T-RAM and Monitor Area, it must be a virtual address.

The action taken is determined by value of M2, whose format is shown below.



RRCB Reference and Change Table Functions

SEL=0 selects RCT functions. (The RCT is described in Section 4.3.8.) When SEL=0, the operand 1 address is treated as a physical address.

When SEL=0 and RCT=0, the reference and change (RC) bits associated with this page frame are fetched and moved to the condition code bits of the PCW.

When SEL=0 and RCT=1, the RC bits are cleared to zero after being fetched; the condition code reflects the state of the RC bits before they were cleared.

Resulting Condition Code

0 Reference bit 0, change bit 0
1 Reference bit 0, change bit 1
2 Reference bit 1, change bit 0
3 Reference bit 1, change bit 1

Program Exceptions

Access (addressing only, operand 1)
Privileged operation

Monitor Area Functions

SEL=1 selects T-RAM and Monitor Area functions. (These structures are described, respectively, in Sections 4.3.3 and 4.3.7.) When SEL=1, the operand 1 address is treated as a virtual address.

When SEL=1 and ALL=1, all T-RAM entries are cleared; i.e, the high-order, fault bit of all entries is set to 1. The Monitor Area is also cleared. This function is available primarily for use during system initialization.

When SEL=1 and ONE=1, the T-RAM entry associated with the (virtual) operand address is cleared; i.e., the fault bit is set to 1.

When SEL=1 and MON=1, those T-RAM entries referenced by the Monitor Area are cleared; the Monitor area is also cleared. For VS100 class systems, whose Monitor area consists of two lists, both lists and their associated T-RAM entries are cleared.

Resulting Condition Code

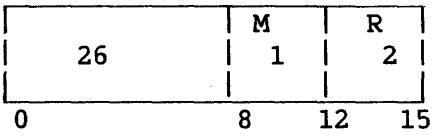
The condition code remains unchanged.

Program Exceptions

Access (addressing only, operand 1)
Privileged operation

RETURN AND POP ON CONDITION (RPC)

RPC M1,R2 (RR)



This instruction is identical to the RTC instruction, except that after all other processing is completed, the stack pointer (General Register 15) will be loaded with the value that was contained in the R2 register before execution of this instruction began. The high-order byte of General Register 15 is set to 0 by this instruction.

Resulting Condition Code

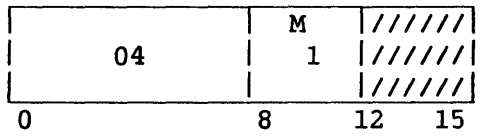
Set with value from stack

Program Exceptions

Access (fetch, bytes popped from the system stack)

Specification (if the current Control Register 1 value is not a multiple of 4)

RTC M1 (RR)



If the state of the condition code is as specified by M1, this instruction locates the most recently-built JSCI save area, which is addressed by Control Register 1. The 3-bit process level field in the save area is compared with the 3-bit process level field in the current PCW. Depending on the outcome of this comparison, one of the following occurs:

1. If the value of the save area process level field is the greater of the two, a program exception (x'06) is taken; a return cannot be made to the more privileged routine.

2. If the values of the two process level fields are equal, the instruction sets General Register 15 (the stack pointer) equal to the current value in Control Register 1 plus 4 (thus skipping over the contents of Register 0 stored the save area by a JSCI instruction); the instruction then pops off the system stack General Registers 1 to 14. The high-order byte of Control Register 1 is set to 0. The next word is popped off the stack and its value replaces the current contents of Control Register 1. One more word is then popped off the stack and used as the program mask byte and address portion of the current PCW. Control then passes to the address specified in the address portion of the PCW.

3. If the value of the save area process level field is the lesser of the two, a stack switch occurs: The contents of General Register 15, Control Register 2, and Control Register 1 are saved in the Stack Header Block addressed by Control Register 8. The value of the save area process level field is used to form an offset into the Stack Header Block Table. The address at this offset is loaded into Control Register 8. General Register 15 is loaded with the contents of the first four bytes of the Stack Header Block addressed by Control Register 8; Control Register 2 is loaded with the contents of the next four bytes in the same SHB. The three low-order bytes of General Register 15 and of Control Register 2 now address, respectively, the top and limit of the active system stack.

Status information is popped from the stack, as already described in paragraph 2. The area of T-RAM monitored by the executive list is cleared if the value of the save area process level field is 0. (This clearing of T-RAM entries takes place only on VS15 and VS100 class systems, as explained in Section 4.3.7.) The value of the PCW process level field is set to that of the save area process level field. Control then passes to the address specified in the address portion of the PCW. (For a general description of stack switching, refer to Section 3.10.)

If the state of the condition code is not as specified by M1, none of the above occurs, and normal instruction sequencing proceeds with the updated instruction address.

Resulting Condition Code

Set with value from stack

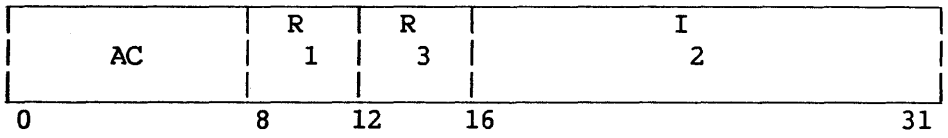
Program Exceptions

Access (fetch, bytes popped from the system stack)

Specification (if the current Control Register 1 value is not a multiple of 4; if the value of the save area process level field exceeds that of the PCW save area process level field). The contents of General Registers 1 to 14 may have been changed by the time the specification exception is taken. However, General Register 15 and Control Register 1 will not be changed.

SAVE THEN 'AND' SYSTEM MASK (STNSM)[P]

STNSM R1,R3,I2 (RS)



This instruction tests whether to save the current PCW status field. If the R1 field of the instruction is 0, the saving is bypassed. If R1 is not 0, the current PCW status field is saved in Bits 16-31 of register R1. Bits 0-15 of R1 are unchanged.

After the saving is performed or bypassed, the R3 field of the instruction is tested. If it is 0, I2 is ANDed with the current PCW status field. If R3 is not 0, I2 is ANDed with Bits 16-31 of register R3, and this replaces the current PCW status field.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Privileged operation

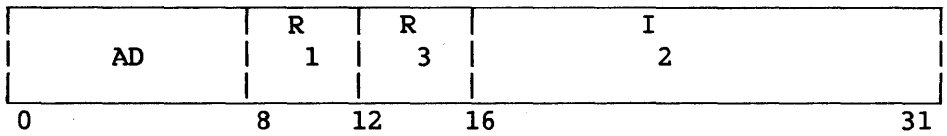
Programming Note

This instruction is normally used for one of three functions:

1. Turning off specified bits in the PCW
2. Turning off specified bits in the PCW while saving the previous status
3. Re-establishing the previous status.

SAVE THEN 'OR' SYSTEM MASK (STOSM)[P]

STOSM R1,R3,I2 (RS)



This instruction first tests whether to save the current PCW status field. If the R1 field of the instruction is 0, the saving is bypassed. If R1 is not 0, the current PCW status field is saved in Bits 16-31 of register R1. Bits 0-15 are unchanged.

After the saving is performed or bypassed, the R3 field of the instruction is tested. If it is 0, I2 is ORed with the current PCW status field. If R3 is not 0, I2 is ORed with Bits 16-31 of register R3 and this replaces the current PCW status field.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Privileged operation

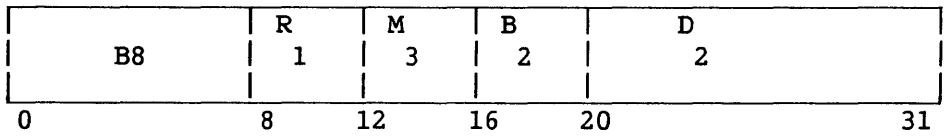
Programming Note

This instruction is normally used for one of three functions:

1. Turning on specified bits in the PCW
2. Turning on specified bits in the PCW while saving the previous status
3. Re-establishing the previous status.

SCAN FOR BYTE (SCAN)

SCAN R1,M3,D2(B2) (RS)



The first operand designates an address-length register pair (general registers R1 and R1+1, with R1 even-numbered). Second operand base and displacement calculations are performed according to the rules for address arithmetic, and the low-order byte of the resulting address is used as an immediate operand. The byte string specified by the first operand address-length register pair is scanned in order of ascending or descending memory addresses, comparing each byte with the second operand byte value, until an equal or unequal value is found.

When the scan is descending, the first byte examined is located at the address of the end of the string as computed by adding the address of the string and its length minus one.

Options are selected by the M3 field as follows:

Bit 0	Ascending scan if 0; descending scan if 1
Bit 1	Stop on equal compare if 0; stop on unequal compare if 1
Bits 2, 3	Must be 0 (specification exception if not)

For an ascending scan, the address-length register pair is updated as follows: Register R1 contains the address of the byte that satisfied the specified condition, or of the first byte beyond the string if the condition is not satisfied. Bits 8-31 of register R1+1 contain either the length of that part of the string including and above the byte on which the condition was satisfied, or 0 if the condition was not satisfied.

For a descending scan, the address-length register pair is updated as follows: Bits 8-31 of register R1 are unchanged. Bits 8-31 of register R1+1 contain either the length of that part of the string including and below the byte on which the condition was satisfied, or 0 if the condition was not satisfied.

The high-order byte (Bits 0-7) of register R1 is set to 0 by the instruction.

The execution of the instruction is interruptible. When an interruption occurs after a unit of operation other than the last one, the contents of registers R1 and R1+1 are incremented and/or decremented so that the instruction, when re-executed, resumes at the point of interruption. The instruction may be refetched from main storage even in the absence of an interruption during execution.

Resulting Condition Code

0 Condition not satisfied

1 Condition satisfied, other than at end of operand 1

2 Condition satisfied at end of operand 1 (highest-addressed byte ascending; lowest-addressed byte descending)

3 --

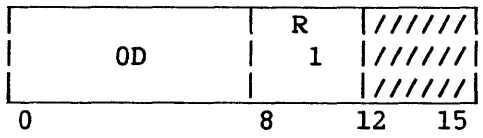
Program Exceptions

Specification
Access (fetch, operand 1)

Programming Notes

For general notes on interruptible instructions, refer to MOVE CHARACTERS LONG.

SPM R1 (RR)



Bits 0-7 of the general register specified by the R1 field replace the condition code and the rest of the program mask bits of the current PCW. Bits 8-31 of the register specified by the R1 field are ignored. The contents of the register specified by the R1 field remain unchanged.

The instruction permits setting of the condition code and the rest of the program mask bits in either the problem program or the supervisor state.

Resulting Condition Code

The code is set according to Bits 0-1 of the register specified by R1.

Program Exception

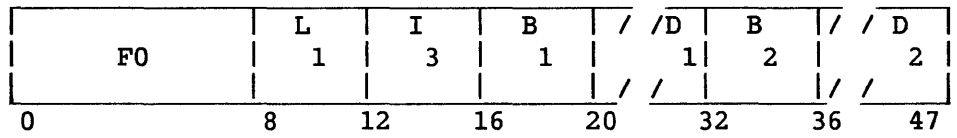
None

Programming Note

Bits 0-7 of the general register may have been loaded from the PCW by BRANCH AND LINK (BAL).

SHIFT AND ROUND DECIMAL (SRP)

SRP D1(L1,B1),D2(B2),I3 (SS)



The first operand is shifted in the direction and for the number of digit positions specified by the second operand address. When shifting to the right is specified, the first operand is rounded by the rounding factor, I3. L1 is the operand length, minus 1.

The second operand address is not used to designate data; instead, the contents of bit positions 26-31 of the address are considered a signed fixed-point quantity, indicating the direction of the shift and the number of digit positions to be shifted. The remainder of the address is ignored. When Bit 26 of the second operand address is 0, a left-shift is specified, and Bits 27-31 of the address are considered a true binary number specifying the number of digit positions of shift. When Bit 26 is 1, a right-shift is specified, and Bits 27-31, considered as a binary number in 2's-complement notation, specify the amount of the shift.

The first operand is considered to be in the packed decimal format and is checked for the validity of decimal digit codes. Only its digit portion is shifted; the sign position does not participate in the shifting. Zeros are supplied for the vacated digit positions. The validity of the first operand is checked and the condition code is set even if a shift amount of 0 is specified. A result of 0 is made positive.

If a significant digit is shifted out of the high-order digit position during left-shift, a decimal overflow condition is recognized. The operation is completed by ignoring the overflow.

During right-shift, bit positions 12-15, the contents of the I3 field, are used as a rounding factor. The shifted operand is rounded by decimally adding the rounding factor to the last digit shifted out and propagating the carry, if any, to the left. Both the first operand and the rounding factor are considered positive quantities for the purpose of this addition. Except for validity checking and the participation in rounding, the digits shifted out of the low-order digit position are ignored and lost. The validity of the rounding-factor code is checked regardless of the direction and amount of shift specified.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 Result overflows

Program Exceptions

Access (fetch and store, operand 1)
Data
Decimal overflow

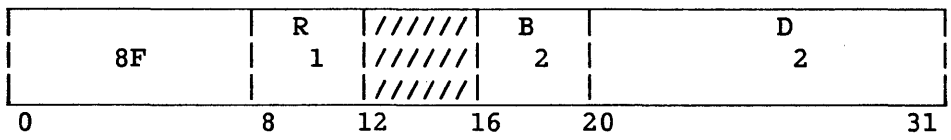
Programming Note

Because the 2's-complement notation is employed, SHIFT AND ROUND DECIMAL can be used for shifting up to 31 digit positions left and up to 32 digit positions right. This is sufficient to clear all digits of any decimal field even when rounding in right-shift is specified.

Please refer to the programming note for SLDA.

SHIFT LEFT DOUBLE (SLDA)

SLDA R1,D2(B2) (RS)



The double-length integer part of the first operand is shifted left the number of bits specified by the second operand address. Bits 12-15 of the instruction are ignored.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The R1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R1 is odd.

The first operand is treated as a number with 63 integer bits and a sign in the sign position of the high-order register. The sign remains unchanged. The high-order bit position of the R1+1 register contains an integer bit, and the contents of the R1+1 register participate in the shift in the same manner as the other integer bits. Zeros are supplied to the vacated positions of the registers.

If a bit unlike the sign bit is shifted out of bit position 1 of the R1 register, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is set to 1.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 Overflow

Program Exceptions

Fixed-point overflow
Specification

Programming Notes

The eight shift instructions provide the following three pairs of alternatives: left or right, single or double, and algebraic or logical. Algebraic shifts differ from the logical shifts in that overflow is recognized, the condition code is set, and the high-order bit participates as a sign in algebraic shifts.

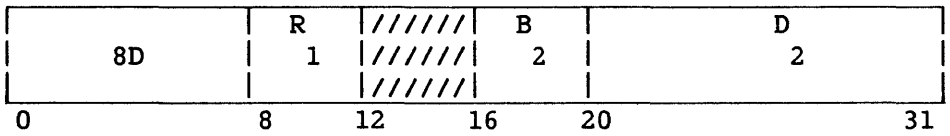
The maximum shift amount that can be specified is 63. For algebraic shifts this is sufficient to shift out the entire integer field. Since 64 bits participate in the double-logical shifts, the entire register contents cannot be shifted out.

A shift amount of 0 in the two algebraic double-shift operations provides a double-length sign and magnitude test.

The base register participating in the generation of the second operand address permits indirect specification of the shift amount. A 0 in the B2 field indicates the absence of indirect shift specification.

SHIFT LEFT DOUBLE LOGICAL (SLDL)

SLDL R1,D2(B2) (RS)



The double-length first operand is shifted left the number of bits specified by the second operand address. The second operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The R1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R1 is odd.

All 64 bits of the register pair specified by R1 participate in the shift. Most significant bits are shifted out of the first register and are lost. Zeros are supplied to the vacated positions of the registers.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

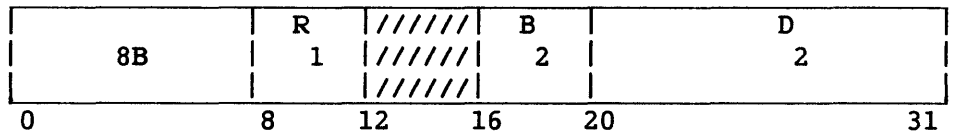
Specification

Programming Note

Please refer to the programming notes for SLDA.

SHIFT LEFT SINGLE (SLA)

SLA R1,D2(B2) (RS)



The integer part of the first operand is shifted left the number of bits specified by the second operand address. Bits 12-15 of the instruction are ignored.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the left-shift. Zeros are supplied to the vacated low-order register positions.

If a bit unlike the sign bit is shifted out of position 1, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is set to 1.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 Overflow

Program Exceptions

Fixed-point overflow

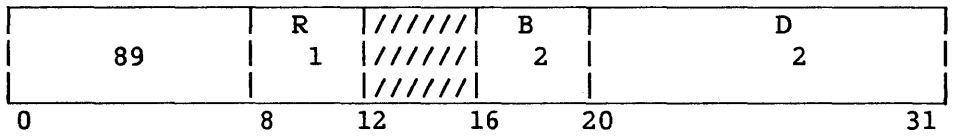
Programming Notes

For numbers with an absolute value of less than 2**30, a left shift of one bit position is equivalent to multiplying the number by 2.

Please refer to the programming notes for SLDA.

SHIFT LEFT SINGLE LOGICAL (SLL)

SLL R1,D2(B2) (RS)



The first operand is shifted left the number of bits specified by the second operand address.

The second operand address is not used to address data; its least significant six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 32 bits of the general register specified by R1 participate in the shift. Most significant bits are shifted out and are lost. Zeros are supplied to the vacated least-significant register positions.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

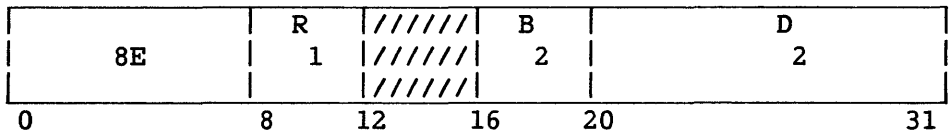
None

Programming Note

Please refer to the programming notes for SLDA.

SHIFT RIGHT DOUBLE (SRDA)

SRDA R1,D2(B2) (RS)



The double-length integer part of the first operand is shifted right the number of places specified by the second operand address. Bits 12-15 of the instruction are ignored.

The R1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R1 is odd.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a number with 63 integer bits and a sign in the sign position of the high-order register. The sign remains unchanged. The high-order bit position of the low-order register contains an integer bit, and the contents of the low-order register participate in the shift in the same manner as the other integer bits. The low-order bits are shifted out without inspection and are lost. Bits equal to the sign are supplied to the vacated positions of the registers.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 --

Program Exceptions

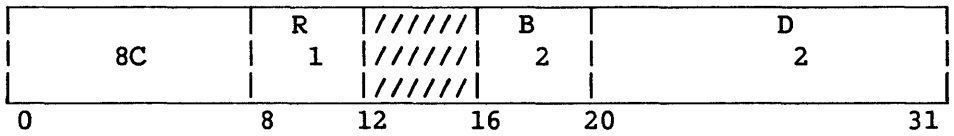
Specification

Programming Note

Please refer to the programming notes for SLDA.

SHIFT RIGHT DOUBLE LOGICAL (SRDL)

SRDL R1,D2(B2) (RS)



The double-length first operand is shifted right the number of bits specified by the second operand address.

The R1 field of the instruction specifies an even/odd pair of registers and must contain an even register address. A specification exception occurs when R1 is odd.

The second operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the register pair specified by R1 participate in the shift. Least significant bits are shifted out of the second register and are lost. Zeros are supplied to the vacated positions of the registers.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

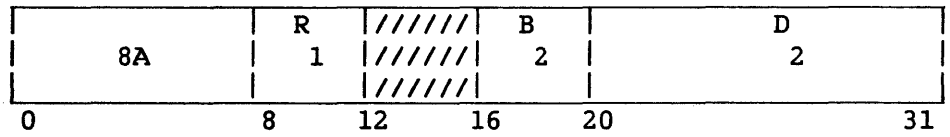
Specification

Programming Note

Please refer to the programming notes for SLDA.

SHIFT RIGHT SINGLE (SRA)

SRA R1,D2(B2) (RS)



The integer part of the first operand is shifted right the number of bits specified by the second operand address. Bits 12-15 of the instruction are ignored.

The second operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the right-shift. Bits equal to the sign are supplied to the vacated high-order bit positions. Low-order bits are shifted out without inspection and are lost.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 --

Program Exceptions

None

Programming Notes

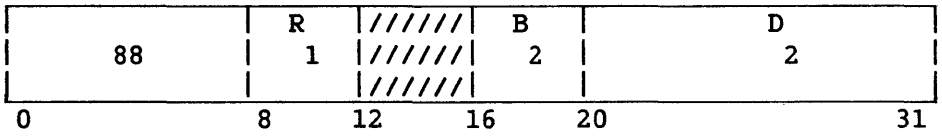
A right-shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the value of the field is that obtained by dividing the value by 2. When an odd number is shifted right one position, the value of the field is that obtained by dividing the next lower number by 2.

Shifts of from 31 to 63 bit positions cause the entire integer to be shifted out of the register. When the entire integer field of a positive number has been shifted out, the register contains a value of 0. For a negative number, the register contains a value of -1.

Please refer to the programming notes for SLDA.

SHIFT RIGHT SINGLE LOGICAL (SRL)

SRL R1,D2(B2) (RS)



The first operand is shifted right the number of bits specified by the second operand address.

The second operand address is not used to address data; its least significant six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 32 bits of the general register specified by R1 participate in the shift. Least significant bits are shifted out and are lost. Zeros are supplied to the vacated most-significant register positions.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

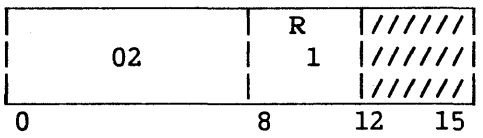
None

Programming Note

Please refer to the programming notes for SLDA.

START I/O (SIO)[P]

SIO R1 (RR)



An I/O command is initiated at the addressed I/O device. The instruction START I/O is executed only when the system is in the supervisor state.

Bits 15 to 31 of R1 identify the device to which the instruction applies. Bits 0 to 15 are ignored. Device address formats for VS systems are described in Section 9.7.

Before the instruction is issued, the command table address (CTA) must be stored in the appropriate entry of the I/O status table, and the I/O command word (IOCW) must be stored in the appropriate entry of the I/O command table (IOCT) addressed by the CTA. These structures and their use by the SIO instruction are described in Chapter 9.

During instruction execution, the I/O operation defined by the prestored IOCW is presented to the BP, BA, or IOC addressed by the PDA. These processors may accept or reject the command, as explained in Section 9.13. Acceptance of the command is indicated by a zero condition code; rejection, by a nonzero condition code.

It is possible that having been accepted by the addressed BP, BA, or IOC, the I/O command is later rejected because of a condition at the addressed device, or because the addressed device is nonexistent. This rejection is reported in the Status Qualifier Byte (SQB), an extension of the IOSW. The SQB format is described in Section 9.6; types of rejection reported in the SQB are described in Section 9.15.

When an SIO instruction is completed with a condition code of 0, the I/O command is usually carried out; exceptions are noted in the preceding paragraph. A pending I/O interruption is established on completion of the operation. Until the completion interrupt has been received, the IOCW must not be changed. The IOCW is not changed by the I/O processors.

Resulting Condition Code

	VS15, VS65	VS100	VS300
0	Successful	Successful	Successful
1	Not used	Not used	Not used
2	Not used	IOP busy	Not used
3	BP busy	IPC-IN busy	IOC busy or nonexistent

Program Exceptions

Privileged operation

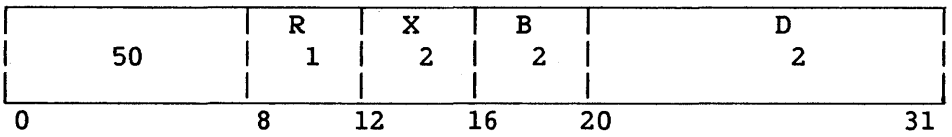
Programming Notes

The completion of the I/O operation initiated by the SIO is indicated by an I/O interruption. Looping on an SIO instruction should be avoided since it may interfere with the operation of the IOP.

Telecommunications (TC) IOPs use the SIO instruction rather than CIO for memory diagnostic operations.

STORE (ST)

ST R1,D2(X2,B2) (RX)



The first operand is stored at the second operand location. The second operand must be four bytes long, and it requires fullword alignment.

The 32 bits in the general register are placed unchanged at the second operand location.

Resulting Condition Code

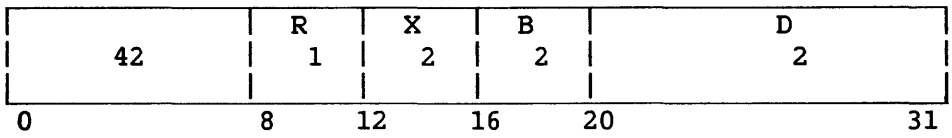
The condition code remains unchanged.

Program Exceptions

Access (store, operand 2)
Specification

STORE CHARACTER (STC)

STC R1,D2(X2,B2) (RX)



Bit positions 24-31 of the general register designated as the first operand are placed at the second operand address. The byte to be stored is not changed or inspected.

Resulting Condition Code

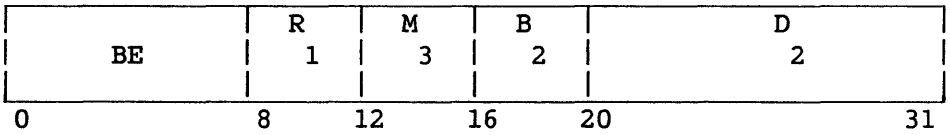
The condition code remains unchanged.

Program Exceptions

Access (store, operand 2)

STORE CHARACTERS UNDER MASK (STCM)

STCM R1,M3,D2(B2) (RS)



Bytes selected from the first operand under control of a mask are placed in contiguous byte locations beginning at the second operand address.

The contents of the M3 field, bit positions 12-15, are used as a mask. The four bits of the mask, left to right, correspond one for one with the four bytes, left to right, of the general register designated by the R1 field. The bytes corresponding to 1s in the mask are placed in the same order in successive and contiguous memory locations beginning with the location designated by the second operand address. The number of bytes stored is equal to the number of 1s in the mask. The contents of the general register remain unchanged.

When the mask is not 0, exceptions associated with storage-operand access are recognized only for the number of bytes specified by the mask. When the mask is 0, access exceptions are recognized for one byte.

Resulting Condition Code

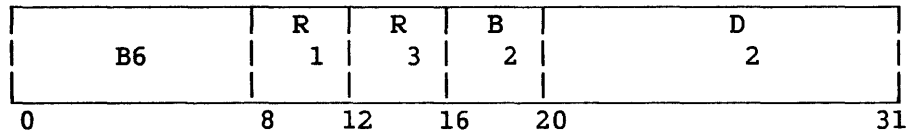
The condition code remains unchanged.

Program Exceptions

Access (store, operand 2)

STORE CONTROL (STCTL)

STCTL R1,R3,D2(B2) (RS)



The set of control registers starting with the control register designated by the R1 field and ending with the one designated by the R3 field is stored at the locations designated by the second operand address.

The memory area where the contents of the control registers are placed starts at the location designated by the second operand address and continues through as many memory words as the number of control registers specified. The contents of the control registers are stored in ascending order of their addresses, starting with the control register designated by the R1 field and continuing up to and including the control register designated by the R3 field. The contents of the control registers remain unchanged.

Whenever the memory reference causes an access exception, the exception is indicated. The second operand must be designated on a word boundary; otherwise, a specification exception is recognized, and the operation is suppressed. A specification exception will also be recognized if R1 is numbered higher than R3.

Resulting Condition Code

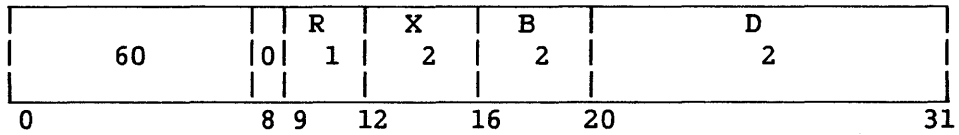
The condition code remains unchanged.

Program Exceptions

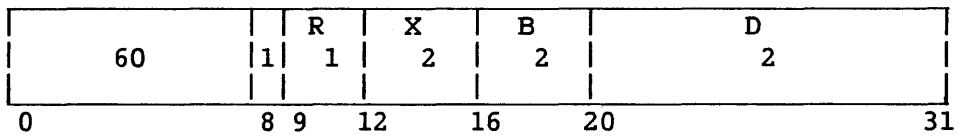
Access (store, operand 2)
Specification

STORE (FLOATING-POINT) (STD, STE)

STD R1,D2(X2,B2) (RX, Long)



STE R1,D2(X2,B2) (RX, Short)



The first operand is stored at the second operand location. The first operand, a floating-point register, remains unchanged. For STD, the second operand must be eight bytes in length and fullword aligned. For STE, the second operand must be four bytes in length and fullword aligned.

Resulting Condition Code

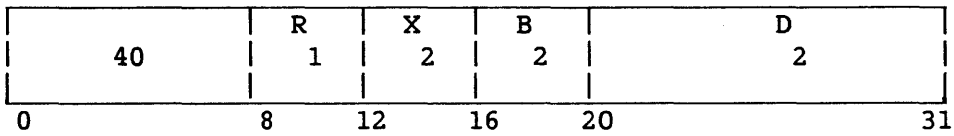
The condition code remains unchanged.

Program Exceptions

Addressing
Protection (store violation)
Specification

STORE HALFWORD (STH)

STH R1,D2(X2,B2) (RX)



The contents of bit positions 16-31 of the general register designated by the R1 field are placed unchanged at the second operand location. The second operand is two bytes in length and requires halfword alignment.

Resulting Condition Code

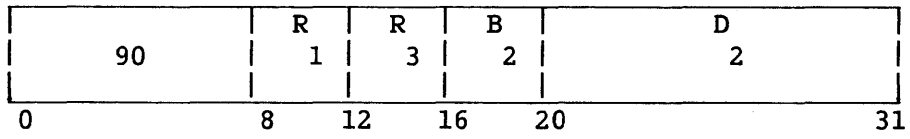
The condition code remains unchanged.

Program Exceptions

Access (store, operand 2)
Specification

STORE MULTIPLE (STM)

```
STM     R1,R3,D2(B2)              (RS)
```



The set of general registers starting with the register specified by R1 and ending with the register specified by R3 is stored at the locations designated by the second operand address.

The memory area where the contents of the general registers are placed starts at the location designated by the second operand address and continues through as many words as needed.

The general registers are stored in the ascending order of their addresses, starting with the register specified by R1 and continuing up to and including the register specified by R3, with Register 0 following Register 15. The contents of the general registers remain unchanged.

Operand 2 requires fullword alignment.

Resulting Condition Code

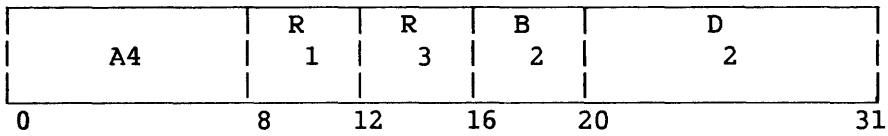
The condition code remains unchanged.

Program Exceptions

Access (store, operand 2)
Specification

STORE SEGMENT CONTROL REGISTER (STSCTL)[P]

STSCTL R1,R3,D2(B2) (RS)



The set of segment control registers (SCRs) starting with the register specified by R1 and ending with the register specified by R3 is stored at the location designated by the second operand address. Each segment control register requires eight bytes of data from memory. Allowable SCR values are 0, 2, 4, and 6.

The memory area where the contents of the SCRs are placed starts at the location designated by the second operand address and continues through as many words as needed.

The contents of the SCRs are stored in ascending order of their addresses, starting with the register specified by R1 and continuing up to and including the register specified by R3. R3 must be greater than or equal to R1. The contents of the SCRs remain unchanged.

Operand 2 requires fullword alignment.

Resulting Condition Code

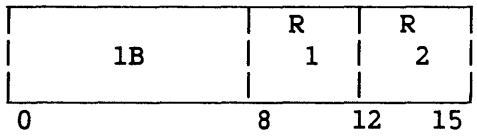
The condition code remains unchanged.

Program Exceptions

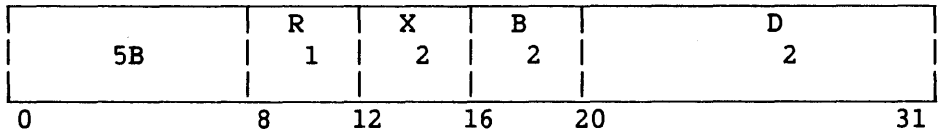
Access (store, operand 2)
Privileged operation
Specification

SUBTRACT (SR, S)

SR R1,R2 (RR)



S R1,D2(X2,B2) (RX)



The second operand, which must be fullword aligned, is subtracted from the first operand, and the difference is placed in the first operand location.

Subtraction is considered to be performed by adding the 1's complement of the second operand and a low-order 1 to the first operand. All 32 bits of both operands participate, as in ADD. If the carry from the sign-bit position and the carry from the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is set to 1.

Resulting Condition Code

0 Difference is 0
1 Difference is less than 0
2 Difference is greater than 0
3 Overflow

Program Exceptions

Specification
Access (fetch, operand 2 of S only)
Fixed-point overflow

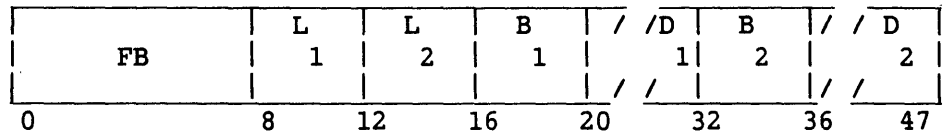
Programming Notes

The use of the 1's complement and the low-order 1 instead of the 2's complement of the second operand is necessary for proper recognition of overflow when the maximum negative number is subtracted.

When in the RR format the R1 and R2 fields designate the same register, subtracting is equivalent to clearing the register.

Subtracting a maximum negative number from another maximum negative number gives a result of 0 and no overflow.

SP D1(L1,B1),D2(L2,B2) (SS)



The second operand is subtracted from the first operand, and the difference is placed in the first operand location.

Subtraction is algebraic, taking into account the signs and all digits of both operands. SUBTRACT DECIMAL is similar to ADD DECIMAL, except that the sign of the second operand is changed from positive to negative or from negative to positive after the operand is obtained from memory and before the arithmetic is performed.

The sign of the difference is determined by the rules of algebra.

L1 and L2 are the operand lengths in bytes, minus 1.

Resulting Condition Code

0 Difference is 0
1 Difference is less than 0
2 Difference is geater than 0
3 Overflow

Program Exceptions

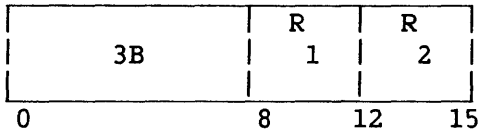
Access (fetch, operand 2; store, operand 1)
Data
Decimal overflow

Programming Note

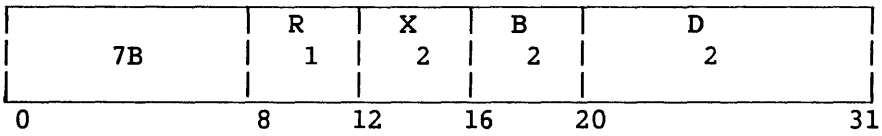
The operands of SUBTRACT DECIMAL may overlap when their least significant bytes coincide, even when their lengths are unequal. This property may be used to set to 0 an entire field or the least significant part of a field.

SUBTRACT DECIMAL (FLOATING-POINT) (SQR, SQ)

SQR R1,R2 (RR)



SQ R1,D2(X2,B2) (RX)



The second operand is subtracted from the first operand, and the normalized difference is placed in the first operand location. Fullword alignment is required.

The SUBTRACT DECIMAL (FLOATING-POINT) instruction is similar to ADD DECIMAL (FLOATING-POINT), except that the sign of the second operand is inverted before addition.

The sign of the difference is derived by the rules of algebra. The sign of a difference with zero result fraction is always positive.

Resulting Condition Code

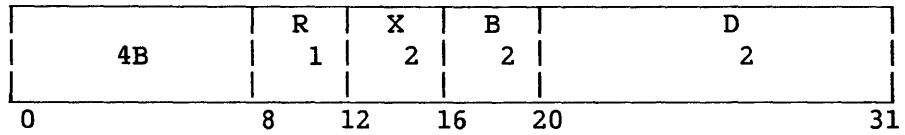
0 Result fraction is 0
1 Result fraction is less than 0
2 Result fraction is greater than 0

Program Exceptions

Specification
Data
Significance
Exponent overflow
Exponent underflow
Access (SQ only)

SUBTRACT HALFWORD (SH)

SH R1,D2,(X2,B2) (RX)



The second operand is subtracted from the first operand, and the difference is placed in the first operand location. The second operand is two bytes in length, must be halfword aligned, and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the subtraction by propagating the sign-bit value through the 16 high-order bit positions.

Subtraction is considered to be performed by adding the 1's complement of the expanded second operand and a low-order 1 to the first operand. All 32 bits of both operands participate, as in ADD. If the carry from the sign-bit position and the carry from the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow occurs. The overflow causes a program interruption when the fixed-point overflow mask bit is 1.

Resulting Condition Code

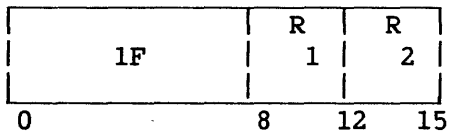
0 Difference is 0
1 Difference is less than 0
2 Difference is greater than 0
3 Overflow

Program Exceptions

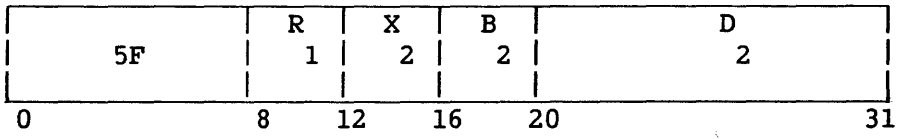
Access (fetch, operand 2)
Fixed-point overflow
Specification

SUBTRACT LOGICAL (SLR, SL)

SLR R1,R2 (RR)



SL R1,D2(X2,B2) (RX)



The second operand is subtracted from the first operand, and the difference is placed in the first operand location. The occurrence of a carry from the sign position is recorded in the condition code.

Logical subtraction is considered to be performed by adding the 1's complement of the second operand and a low-order 1 to the first operand. All 32 bits of both operands participate, without further change to the resulting leftmost bit position.

If a carry from the sign position occurs, the leftmost bit of the condition code is made 1. In the absence of a carry, the left bit is made 0. When the sum is 0, the rightmost bit of the condition code is made 0. A nonzero sum is indicated by a 1 in the rightmost bit.

The second operand of the SL instruction requires fullword alignment.

Resulting Condition Code

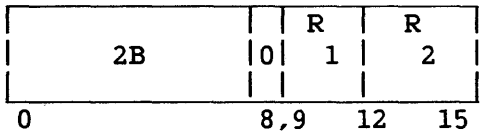
0 --
1 Difference is not 0 (no carry)
2 Difference is 0 (carry)
3 Difference is not 0 (carry)

Program Exceptions

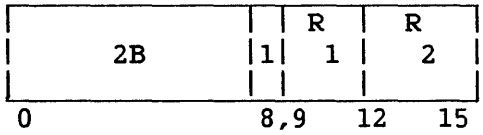
Specification
Access (fetch, operand 2 for SL)

SUBTRACT NORMALIZED (FLOATING-POINT) (SDR, SER, SD, SE)

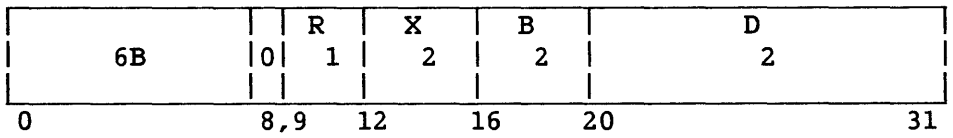
SDR R1,R2 (RR, Long)



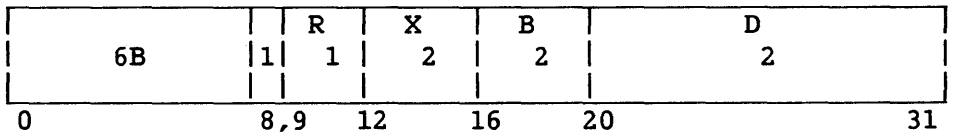
SER R1,R2 (RR, Short)



SD R1,D2(X2,B2 (RX, Long))



SE R1,D2(X2,B2 (RX, Short)



The second operand is subtracted from the first operand, and the normalized difference is placed in the first operand location.

SUBTRACT is similar to ADD NORMALIZED, except that the sign of the second operand is inverted before addition.

The sign of the difference is derived according to the rules of algebra. The sign of a difference with a zero result fraction is always positive.

The second operand of the SD instruction requires fullword alignment and is eight bytes long.

Resulting Condition Code

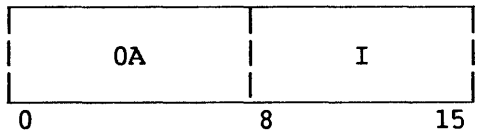
0 Result fraction is 0
1 Result is less than 0
2 Result is greater than 0
3 --

Program Exceptions

Specification
Significance
Exponent overflow
Exponent underflow
Access

SUPERVISOR CALL (SVC)

SVC I (RR)



If the high-order byte of the Supervisor Call New PCW is less than the value in Bits 8-15 of the instruction, the instruction is suppressed with a supervisor call range program exception. Otherwise, the system stack vector is retrieved from General Register 15 and Control Register 2. The stack pointer (Register 15) is decremented by 8. The currently-active PCW is stored in the eight bytes addressed by the decremented stack pointer. The value in Bits 8-15 of the instruction are placed in the interruption code field of the PCW just stored. Then pushed onto the stack is a byte consisting of four high-order zeros, followed by the three PCW process level bits, followed by a binary 1, which indicates a SVC-type save area. Then the three low-order bytes of Control Register 1 are pushed onto the stack, followed by the contents of General Registers 14 to 0. The three low-order bytes of Control Register 1 are then set to the value of the updated stack pointer, with a high-order byte of binary 0s. The high-order word of the Supervisor Call New PCW is then added to four times the contents of bit positions 8 to 15 of the instruction, and the word at the resulting address (which must be the address of a fullword present in main memory, not page faulted) becomes the current PCW address portion. The second word of the Supervisor Call New PCW becomes the current PCW status portion.

Neither SVC or SVCX, alone or paired, can cause a change of process level.

Resulting Condition Code

The condition code is replaced by the condition code in the new PCW.

Program Exceptions

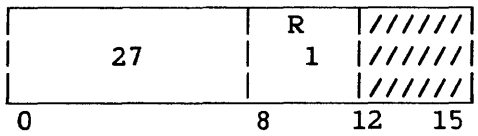
Stack overflow

Access (store, bytes pushed on to stack; fetch, address word to become current PCW address portion)

Specification

Supervisor call range

SVCX (RR)



General Registers 0 through 14 are loaded from the words addressed by Control Register 1. Control register 1 is loaded from the word above these (beginning 60 bytes above the word addressed by Control Register 1). The high-order byte of Control Register 1 is set to binary 0. General Register 15 is loaded with the value in the general register specified by the R1 field of the instruction. The active PCW is replaced by the two words on the system stack starting 64 bytes above the word that had been addressed by Control Register 1 before it was updated.

Resulting Condition Code

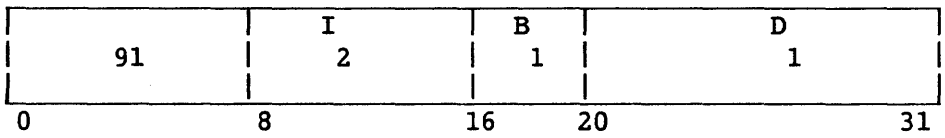
The condition code is replaced by that in the new PCW.

Program Exceptions

Access (fetch, bytes on stack)
Privileged operation
Specification

TEST UNDER MASK (TM)

TM D1(B1),I2 (SI)



The state of the first operand bits selected by a mask is used to set the condition code.

The byte of immediate data, I2, is used as an 8-bit mask. The bits of the mask are made to correspond one for one with the bits of the character in memory specified by the first operand address.

A mask bit of 1 indicates that the memory bit is to be tested. When the mask bit is 0, the memory bit is ignored. When all memory bits thus selected are 0, the condition code is made 0. The condition code is also made 0 when the mask is all 0s. When the selected bits are all 1s, the code is made 3; otherwise, the code is made 1. The character or characters in memory or the registers are not changed.

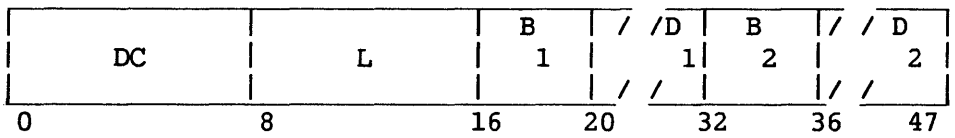
Resulting Condition Code

0 Selected bits all 0; mask is all 0s
1 Selected bits mixed 0s and 1s
2 --
3 Selected bits all 1

Program Exceptions

Access (fetch, operand 1)

TR D1(L,B1),D2(B2) (SS)



The 8-bit bytes of the first operand are used as arguments to reference the list designated by the second operand address. Each function byte selected from the list replaces the corresponding argument in the first operand.

The bytes of the first operand are selected one by one for translation, proceeding left to right. Each argument byte is added to the initial second operand address, in the least significant bit positions. The sum is used as the address of the function byte, which then replaces the original argument byte.

All result data is valid. The operation proceeds until the first operand field is exhausted. The list is not altered unless an overlap occurs.

L is the length of operand 1, minus 1.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

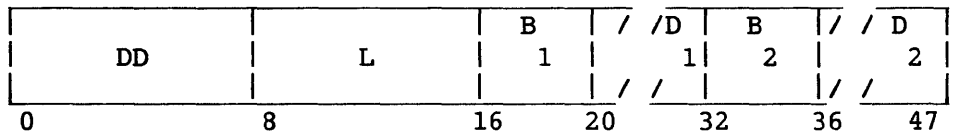
Access (fetch, operands 1 and 2; store, operand 1)

Programming Note

A table length of 256 bytes is recommended as matching the full range of values that an argument byte can assume (i.e., 0 through 255). The starting address of the table must be at least 256 bytes away from the end of the program's legal memory area because the instruction assumes that the table is potentially 256 bytes long. This restriction is easily met by allocating 256 bytes for the table or placing it near the beginning of a program's legal memory.

TRANSLATE AND TEST (TRT)

TRT D1(L,B1),D2(B2) (SS)



The 8-bit bytes of the first operand are used as arguments to reference the list designated by the second operand address.

The L field is the length of the first operand, minus 1.

Each function byte thus selected from the list is used to determine the continuation of the operation. When the function byte is a 0, the operation proceeds by fetching and translating the next argument byte. When the function byte is nonzero, the operation is completed by inserting the related argument address in General Register 1 and by inserting the function byte in General Register 2.

The bytes of the first operand are selected one by one for translation, proceeding from left to right. The first operand remains unchanged in memory. Fetching of the function byte from the list is performed as in TRANSLATE. The function byte retrieved from the list is inspected for the all-zero combination.

When the first operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting condition code 0. The contents of General Registers 1 and 2 remain unchanged.

When a function byte is nonzero, the related argument address is inserted in the low-order 24 bits of General Register 1. This address points to the argument last translated. The high-order eight bits of Register 1 remain unchanged. The function byte is inserted in the low-order eight bits of General Register 2. Bits 0-23 of Register 2 remain unchanged. Condition code 1 is set when one or more argument bytes have not been translated. Condition code 2 is set if the last function byte is nonzero.

Resulting Condition Code

0 All function bytes that have been translated are 0

1 Nonzero function byte found before the first operand field is exhausted; one or more argument bytes have not been translated

2 The last function byte is nonzero

3 --

Program Exceptions

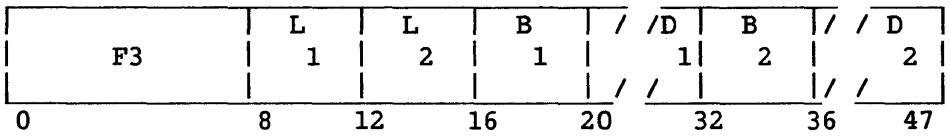
Access (fetch, operands 1 and 2)

Programming Note

The instruction TRANSLATE AND TEST may be used to scan the first operand for characters with special meaning. The second operand, or list, is set up with all-zero function bytes for those characters to be skipped over and with nonzero function bytes for the characters to be detected.

UNPACK (UNPK)

UNPK D1(L1,B1),D2(L2,B2) (SS)



The format of the second operand is changed from packed to zoned form, and the result is placed in the first operand location.

The digits and sign of the packed operand are placed unchanged in the first operand location, using the external format. Zones with coding 0011 are supplied for all bytes except the low-order byte, which receives the sign of the packed operand. The operand digits are not checked for valid codes.

The fields are processed right to left. The second operand is extended with high-order 0s before unpacking, if necessary. If the first operand field is too short to contain all significant digits of the second operand, the remaining high-order digits are ignored. The first and second operand fields may overlap; if so, they are processed by storing the first result byte immediately after the rightmost operand byte is fetched; for the remaining operand bytes, two result bytes are stored immediately after one byte is fetched.

L1 and L2 are the operand lengths, minus 1.

Resulting Condition Code

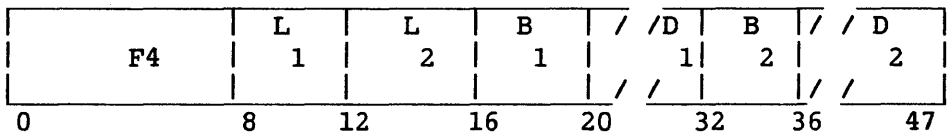
The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2; store, operand 1)

UNPACK UNSIGNED (UNPU)

UNPU D1(L1,B1),D2(L2,B2) (SS)



The format of the second operand is changed from packed to external, and the result is placed in the first operand location.

The digits of the packed operand are converted to ASCII form and are placed in the first operand location. Zones with coding 0011 are supplied for all bytes. The sign of the second operand is ignored. No sign character is supplied in the result.

The fields are processed right to left. The second operand is extended with high-order 0s before unpacking, if necessary. If the first operand field is too short to contain all significant digits of the second operand, the remaining high-order digits are ignored. The first and second operand fields may overlap and are processed by storing the first result byte immediately after the rightmost operand byte is fetched; for the remaining operand bytes, two result bytes are stored immediately after one byte is fetched.

L1 and L2 are the operand lengths, minus 1.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2; store, operand 1)

UNPACK TO EXTERNAL DECIMAL FORMAT (UNPAL)

UNPAL D1(L1,B1),D2(L2,B2) (SS)

DB	L1	L2	B1	D1	B2	D2
0	8	12	16	20	32	36 47

The format of the second operand is changed from packed to character format with a separate trailing sign character, and the result is placed in the first operand location.

The second operand is processed from right to left. First the low-order byte of operand 1 will be filled with either the '+' or '-' character. If the low-order digit position of the last byte of operand 2 is 1101, the character will be '-'; otherwise the character will be '+'. The digits are then moved from operand 2 to operand 1. The digits are copied unchanged from the second operand to the first with a zone of 0011 supplied for each digit.

The digits in the source field are not inspected for valid packed characters and the sign is not inspected for validity.

The fields are processed right to left. The second operand is extended with high-order zero digits before unpacking, if necessary. If the first operand field is too short to contain all significant digits of the second operand, the remaining high-order digits are ignored. The first and second operand fields may overlap, and are processed by storing two result bytes immediately after one byte is fetched.

If the receiving field is one byte, only the sign character will be placed there.

L1 and L2 are the operand lengths, minus 1.

Resulting Condition Code

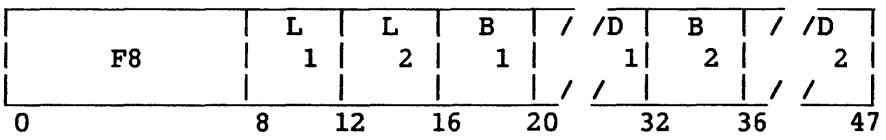
The condition code remains unchanged.

Program Exceptions

Access (fetch, operand 2; store, operand 1)

ZERO AND ADD (ZAP)

ZAP D1(L1,B1),D2(L2,B2) (SS)



The second operand is placed in the first operand location.

The operation is equivalent to an addition to 0. A zero result is positive. When the most significant digits are lost because of overflow, an overflow is recognized. If the decimal overflow mask bit is on when an overflow is recognized, the exception is taken.

Only the second operand is checked for valid sign and digit codes. Extra 0s are supplied if needed on the most significant end. When the first operand field is too short to contain all significant digits of the second operand, the most significant digits are lost and the overflow condition is set. The first and second operand fields may overlap when the rightmost byte of the first operand field is coincident with or to the right of the rightmost byte of the second operand.

L1 and L2 are the operand lengths, minus 1.

Resulting Condition Code

0 Result is 0
1 Result is less than 0
2 Result is greater than 0
3 Overflow

Program Exceptions

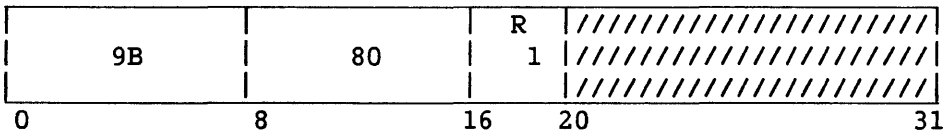
Access (fetch, operand 2; store, operand 1)
Data
Decimal overflow

8.2 EXTENDED OPERATION CODE INSTRUCTIONS

Opcode X'9B' has been designated a 2-byte opcode. Opcodes X'9B00' through X'9B7F' are privileged opcodes; X'9B80' through X'9BFF' are not privileged. Executing an instruction with an undefined opcode in the range from X'9B00' through X'9B7F' while the privileged-instruction trap bit in the PCW is set may result in a privileged-instruction interrupt rather than an invalid-operation interrupt.

STORE CP TYPE AND MICROCODE VERSION (STCPID)

STCPID R1 (S)



A 2-byte code, representing the current CP type and current microcode version, is stored in a general register.

Bits 0-15 of general register R1 are set to 0. The CP type code is stored in Bits 16-23 of register R1; the current microcode version number is stored in Bits 24-31 of register R1.

Current CP type codes are: for the VS15, 5; for the VS65, 7; for the VS100, 4; and for the VS300, 8.

Resulting Condition Code

For the current VS processors, the condition code remains unchanged.

Program Exceptions

None

Programming Note

Bits 0-15 of general register R1, Bits 20-31 of this instruction, and condition code values other than 0 are reserved. They may eventually be used to indicate any optional features present in a particular processor, or for other purposes.

STORE DIAGNOSTIC DATA (STDD)[P]

```
STDD   D1(B1)                (S)

 ┌──────────┬──────────┬──────┬──────────────────┐
 │          │          │  B   │        D         │
 │    9B    │    00    │  1   │        1         │
 │          │          │      │                  │
 └──────────┴──────────┴──────┴──────────────────┘
 0          8          16     20                 31
```

Diagnostic information, including the contents of general and control registers, is stored starting at the location specified by the operand 1 address. This address is not translated. (It is a physical main memory address.) Operand 1 must be fullword aligned, or a specification exception will occur and the instruction will be suppressed.

Diagnostic data is stored in the order shown below. (The parenthetical values in each diagram give the register length in bits.) On VS15 class systems only, floating-point registers, control registers, file registers, and general registers are stored with high-order and low-order halfwords reversed.

VS15

Data Item (and Decimal Size)	Byte Offset from Operand 1 Address
32 File Registers (16)	X'0'
4 Floating-Point Registers (64)	X'40'
Control Registers 0-7 (32)	X'60'
32 CP Work Registers (16)	X'80'
16 General Registers (32)	X'C0
4 Segment Control Registers (64)	X'100'
64 Monitor Area Registers (16)	X'120'
32 Reserved Registers (16)	X'1A0'
Control Registers 8-15 (32)	X'1E0'-X'200'

Data Item (and Decimal Size)	Byte Offset from Operand 1 Address
32 BANK0 File Registers (16)	X'0'
16 General Registers (32)	X'40'
48 BANK1 File Registers (16)	X'80'
4 Floating-Point Registers (64)	X'E0'
32 BANK2 File Registers (16)	X'100'
16 Control Registers (32)	X'140'
18 BANK3 File Registers (16)	X'180'
4 Segment Control Registers (64)	X'1E0'-X'200'

Data Item (and Decimal Size)	Byte Offset from Operand 1 Address
16 General Registers (32)	X'0'
16 Control Registers (32)	X'40'
32 BANK0 File Registers (32)	X'80'
4 Floating-Point Registers (64)	X'100'
4 Segment Control Registers (64)	X'120'
Memory Size	X'140'
Reserved	X'144'
VS Monitor Area Origin	X'178'
VM Monitor Index	X'17C'
32 BANK1 File Registers (32)	X'180'
T-RAM Monitor Area	X'200'-X'400'

Data Item (and Decimal Size)	Byte Offset from Operand 1 Address
16 General Registers (32)	X'0'
16 Control Registers (32)	X'40'
Reserved	X'80'
4 Floating-Point Registers (64)	X'100'
4 Segment Control Registers (64)	X'120'

Resulting Condition Code

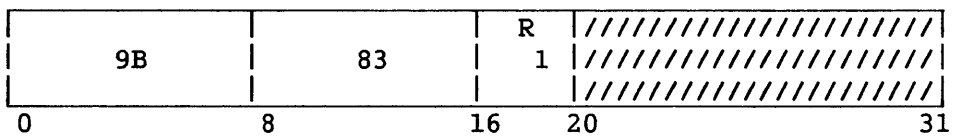
The condition code remains unchanged.

Program Exceptions

Access (store, operand 1)
Specification
Privileged operation

STORE EXTENDED CP TYPE AND MICROCODE VERSION (STLCPID)

STLCPID R1 (S)



Three bytes of information, indicating the CP type, major version of the microcode and minor version, are stored in a general register.

Bits 0-6 of register R1 are reserved for future use. Bit 7 indicates a 16M virtual address when set to 1, an 8M virtual address space when set to 0. The CP type, currently denoted by a single digit, is stored right-justified in Bits 8-15 of register R1; the major microcode version number, consisting of two digits, is stored in Bits 16-23; the minor microcode version number, consisting of two digits, is stored in Bits 24-31.

Identifying numbers for the CP types are: for the VS15, 5; for the VS65, 7; for the VS100, 4; and for the VS300, 8.

Resulting Condition Code

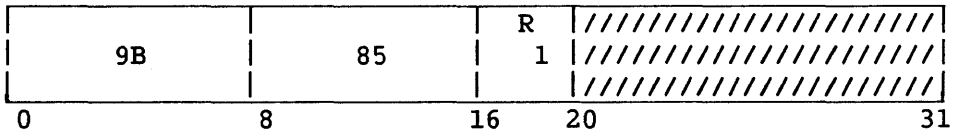
For the current VS processors, the condition code remains unchanged.

Program Exceptions

None

STORE RING NUMBER (STRING)^{P}

STRING R1 (S)



The ring number (i.e., process level) of the current process is stored in a general register.

The ring number, consisting of a single digit, is stored right-justified in general register R1.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

None

CHAPTER 9
INPUT/OUTPUT OPERATION

9.1 INTRODUCTION

This chapter describes the control data that is passed between the CP and I/O processors to control the I/O operations initiated by the START I/O, CONTROL I/O, and HALT I/O machine instructions.

In this chapter, the term "I/O processors" refers to processors that serve as an interface between the CP and device processors. In the context of VS15 and VS65 systems, the term refers to the Bus Processor (BP); in the context of the VS100 system, the term refers to the Bus Adapter (BA) in combination with the IOP; in the context of the VS300 system, the term refers to the I/O Controller (IOC).

Two items of control data, the I/O Command Word (IOCW) and I/O Status words (IOSW) include fields whose contents are device-specific. This chapter describes only those fields of the IOCW and IOSW whose definition is the same for all devices. Device specific fields are described in Chapters 10 through 13.

9.2 SUMMARY OF DATA TRANSFER OPERATIONS

The following outline of a data transfer operation is provided to indicate the functional relationship between the various control data items described later in the chapter. Certain phases of I/O operations are described more fully in Sections 9.14 through 9.18.

To initiate the data transfer operation, the CP executes the privileged START I/O (SIO) instruction. Before execution of this instruction, the operating system has placed an I/O command word (IOCW) in an I/O Command Table (IOCT) that resides in main memory. This word specifies the I/O operation to be performed (e.g., Read, Write), the location of the data to be transferred, the size of the data, and other information. The operating system has also placed in main memory a Command Table Address (CTA), which points to the IOCT.

During execution of the START I/O instruction, the CP presents an I/O request to the I/O processor. On the VS15 and VS65 systems, the CP places in a shared main memory area the physical device address (PDA) of the device involved and an IOCW. On VS100 and and VS300 systems, the CP sends the PDA to an internal register of the I/O processor; using the PDA and CTA, the I/O processor can locate the appropriate IOCW in main memory.

When presented with an I/O request, The I/O processor may be available or busy, and accepts or rejects the request accordingly. After determining the disposition of the request, the CP reports its acceptance or rejection through the condition code of the PCW. There is no further communication between the CP and I/O processor until the I/O Processor interrupts the CP to signal completion of the I/O operation.

Before issuing the interrupt, the I/O processor stores an I/O status word in a status table that starts at main memory Location X'90'. Bit settings in the IOSW indicate successful completion of the I/O operation or an error completion and the nature of that error. An extension of the IOSW, called a Status Qualifier Byte (SQB), may indicate that an I/O request, after being accepted by the I/O processor, was subsequently rejected because the device addressed by the request was not available.

When granting an interrupt, the CP locates the IOSW that has been stored in the I/O status table, and copies the IOSW to main memory Location X'00' for access by the interrupt handler.

NOTE

I/O processors do not perform address translation. Therefore, all addresses passed between the Central and I/O processors must be physical.

9.3 MAIN MEMORY ASSIGNMENTS FOR INTERPROCESSOR COMMUNICATIONS

Table 9-1 summarizes permanent memory assignments for I/O control data:

Table 9-1. Permanent Memory Assignments

Location	Data Item	Written By	System
X'00-07'	IOSW	CP	VS15, VS65, VS100, VS300
X'50'	SQB	CP	VS15, VS65, VS100, VS300
X'80-81'	PDA	CP	VS15, VS65, VS100, VS300
X'82-8F'	(for VS100 and VS300, area reserved for system use; for VS15, VS65: CP-BP communications area)		
X'90' (DAST)	IOSW, SQB CTA	BP OS[a]	VS15, VS65 " "
X'90' (IOPST)	IOSW CTA	IOP OS[a]	VS100 "
X'90' (IOCST)	IOSW, SQB, PDA CTA	IOC OS[a]	VS300 "
per CTA (IOCT)	IOCW	OS[a]	VS15, VS65, VS100, VS300

[a] Operating System

9.3.1 CP-BP Communications Area

As indicated by Table 9-1, Locations X'82-8F' are used on VS15 and VS65 systems for communication between the CP and Bus Processor (BP). The CP-BP communications area is defined in Table 9-2.

Table 9-2. CP-BP Communications Area

Location	Description
X'82'-X'83'	Here the BP pre-stores for the CP the PDA associated with an upcoming interrupt.
X'84-X'8C'	Here the CP stores for the BP any IOCW associated with the current command.
X'8D'	Here the CP stores one of the following command bytes: 00 = Alert 01 = SIO 02 = HIO 03 = CIO
X'8E'-X'8F'	Here the CP stores for the BP the physical address of the device associated with the current command.

9.4 I/O STATUS TABLE

On all VS systems, an I/O status table starting at Location X'90' receives status information on completed I/O operations. On VS15 and VS65 systems, there are as many table entries as supported Device Adapters; on VS100 systems, there are as many table entries as supported IOPs; on the VS300 system, there are as many table entries as supported IOCs.

Table entries are 16 bytes in length. The entry format differs among VS systems, as indicated in the following sections. Common to all formats is a field for the IOSW and for the Command Table Address (CTA).

The I/O processor pre-stores the IOSW in a table entry before signaling an interrupt to the CP. Then the CP, granting the interrupt, copies the IOSW from the table entry to Location X'00'.

The Command Table address (CTA) points to the IOCT associated with the I/O processor specified by the I/O instruction. The operating system places the CTA in the table before a SIO or CIO machine instruction initiates an I/O operation.

9.4.1. VS15, VS65 Device Adapter Status Table (DAST)

The DAST comprises six 16-byte entries -- one entry for each Device Adapter (DA) supported by the system. The entries are arranged in ascending order by DA number. Figure 9-1 shows the format of each entry.

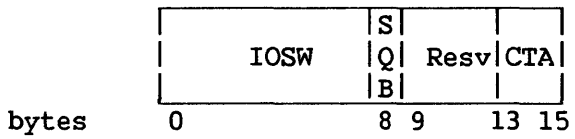


Figure 9-1. DA Status Table (DAST) Entry

The BP pre-stores in the table the Status Qualifier Byte (Byte 8) along with the IOSW. The format and function of the SQB is described in Section 9.6. The CP, when granting an I/O interrupt, copies the IOSW to Location X'00' and the SQB to Location X'50'.

Immediately following the SQB is a four-byte area reserved for system use.

9.4.2 VS100 IOP Status Table (IOPST)

The IOP Status Table (IOPST) comprises sixteen 16-byte entries -- one entry for each IOP supported by the system. The entries are arranged from IOP0-IOP7 on BA1, then from IOP0-IOP7 on BA2. Figure 9-2 shows the format of each entry.

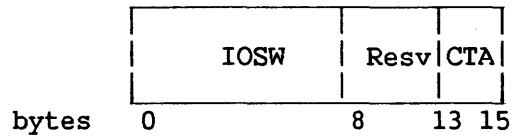


Figure 9-2. IOP Status Table (IOPST) for the VS100

9.4.3 VS300 IOC Status Table (IOCST)

The IOC Status Table (IOCST) comprises fifteen 16-byte entries -- one entry for each IOC supported by the system. The entries are arranged from IOC1-IOC15 on System Bus Interface (SBI) 0. Figure 9-3 shows the format of each entry.

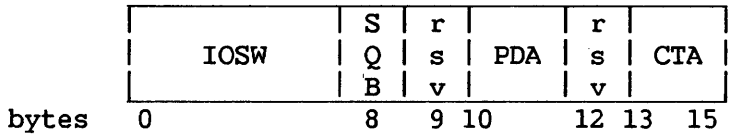


Figure 9-3. IOC Status Table (IOCST) Entry

The SQB (Byte 8) includes an IOSW Active Flag bit, as described in Section 9.6.2

9.5 I/O COMMAND TABLE (IOCT)

There is an IOCT for each I/O processor; it is addressed by the Command Table Address (CTA) field of the I/O status table entry for the same processor. The IOCT may begin at any doubleword-aligned address in main memory, and consists of one 16-byte entry for each device attached to the processor. The entries are arranged from logical device number 0 to the highest logical device number supported by the particular processor.

Each IOCT entry begins with a 9-byte IOCW for the particular device. The next four bytes are reserved for use by the operating system. The operating system uses Bytes 13-15 for the Unit Control Block Address (UCBA) for the device, thereby establishing a direct link between the device address and the device's UCB.

Figure 9-4 shows the format of each IOCT entry.

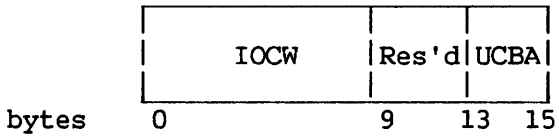


Figure 9-4. I/O Command Table (IOCT)

9.6 STATUS QUALIFIER BYTE (SQB)

An I/O command initiated by the SIO, CIO, or HIO instructions can be accepted or rejected at two stages: 1) When presented by the CP to the I/O processor, or 2) When the I/O processor passes the order to the device specified in the instruction.

The CP reports on the I/O command's initial acceptance or rejection (Stage 1) through setting the condition code for the I/O instruction. Having set the condition code, the CP executes the next instruction; it does not wait to determine whether the I/O command has been finally accepted (Stage 2) before setting the condition code.

The I/O processor reports any later rejection of the I/O command (Stage 2) in the Status Qualifier Byte (SQB) that it writes along with every IOSW. For VS15, VS65, and VS100 systems, a nonzero SQB indicates rejection of the last I/O command directed to the device. On the VS300, a SQB value other than X'80' indicates rejection.

The following sections describe the format and bit definitions of SQBs. Sections 9.14 and 9.15 describe in more detail the conditions under which I/O commands are accepted and rejected.

9.6.1 VS15, VS65, VS100 Status Qualifier Byte

Figure 9-5 shows the format of the SQB for VS15, VS65, and VS100 systems.

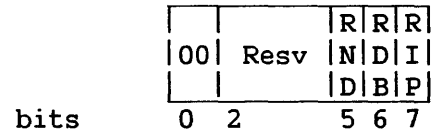


Figure 9-5. Status Qualifier Byte (SQB)

Bits 5-7, when set to one, have the following significance:

Bit 5 (RND) Rejected -- nonexistent device. A SIO or CIO command has been rejected because the device specified by the command is nonexistent.

Bit 6 (RDB) Rejected -- device busy. A CIO or SIO command has been rejected because the particular device is still executing a previous command. The RDB bit is not set after the rejection of a HIO instruction.

Bit 7 (RIP) Rejected -- interrupt pending. A CIO or SIO command has been rejected because the particular device has an unsolicited interrupt to report but the I/O processor has not yet raised its request line (i.e., has not yet stored the unsolicited IOSW in the I/O status table). An HIO instruction is ignored in this case.

9.6.2 V300 Status Qualifier Byte

Figure 9-6 shows the format of the SQB for the VS300 system.

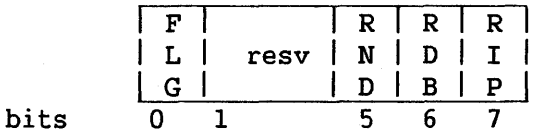


Figure 9-6. VS300 Status Qualifier Byte (SQB)

Bit 0 of the SQB is the IOSW Active Flag. When pre-storing the IOSW and SQB in the IOC Status Table (IOCST), the IOC sets the flag bit to 1; then signals an interrupt. The CP, processing the interrupt, scans the IOCST for an entry whose flag bit is set to one. Having located that entry, the CP sets the flag bit to 0, then copies the IOSW and SQB stored there to main memory Locations X'00' and X'50', respectively.

Section 9.6.1 describes the significance of Bits 5-7 when they are set to 1.

9.7 PHYSICAL DEVICE ADDRESS (PDA)

The general register specified in a SIO, CIO, or HIO instruction holds a 16-bit physical address of the device involved in the I/O operation. The CP passes this physical device address (PDA) to the I/O processor when initiating an I/O operation; the I/O processor passes the PDA to the CP when signalling an interrupt. Because of variations in VS system architecture, there are several forms of the PDA, each described in a following section.

9.7.1 VS15, VS65 PDA

On VS15 and VS65 systems, the PDA is defined as shown in Figure 9-7.

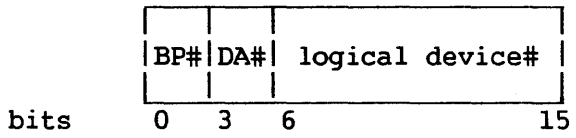


Figure 9-7. VS15, VS65 PDA

The only value allowed for Bits 0-2, which specify the Bus Processor (BP), is '001'.

Bits 3-5 specify one of seven Device Adapters (DAs). The value '000' must be specified for the diskette DA; '010' for the serial device DA supporting Workstation 0. The remaining valid values can be assigned to any type of DA.

Bits 6-15 specify the logical device number (LDN) on the DA. Current DAs do not support the potential 10-bit range of LDNs, and reject instructions containing LDNs that are out of range.

9.7.2 VS100 PDA

On the VS100 system, the PDA is defined as shown in Figure 9-8.

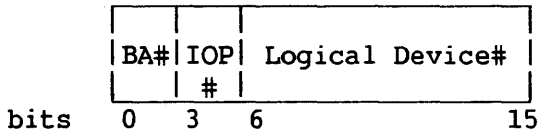


Figure 9-8. VS100 PDA

The values allowed for Bits 0-2, which specify the Bus Adapter (BA) are '001' (BA1) or '010' (BA2).

Bits 3-5 specify the IOP, in the range '000' (IOP0) to '111' (IOP7); the BA interprets Bits 3-5 as indicative of IOP position on the outboard bus.

Bits 6-15 specify the logical device number (LDN) on the IOP. Current IOPs do not support the potential 10-bit range of LDNs, and reject instructions containing LDNs that are out of range.

9.7.3 VS300 PDA

On the VS300 system, the PDA is defined as shown in Figure 9-9.

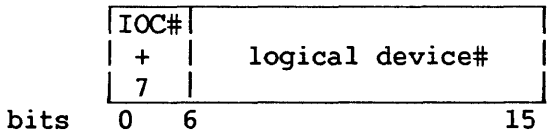


Figure 9-9. VS300 PDA

There is no field that specifies a System Bus Interface (SBI) number, because SBI0 is assumed.

Bits 0-5 specify the I/O Controller (IOC) number (starting at 1) plus 7. Allowable values for this field are 8-22.

Bits 6-15 specify the logical device number (LDN) on the IOC. Current IOCs do not support the potential 10-bit range of LDNs, and reject instructions containing LDNs that are out of range.

9.8 I/O COMMAND WORD (IOCW) FOR SIO INSTRUCTION

The IOCW specifies the command to be executed. For SIO commands initiating data transfer, it designates the storage area associated with the operation. The IOCW is contained in the IOCT entry for the device specified in the SIO instruction.

NOTE

From the time an SIO is accepted until the clearing of the resulting I/O completion interrupt, the IOCW must not be changed. Neither the device nor the I/O processor will change the IOCW.

The IOCW consists of a 6-byte general section and a variable-length device-dependent section, as shown in Figure 9-10. The device-dependent section can be of any length up to three bytes, but is fixed for each device. The IOCW must be fullword-aligned.

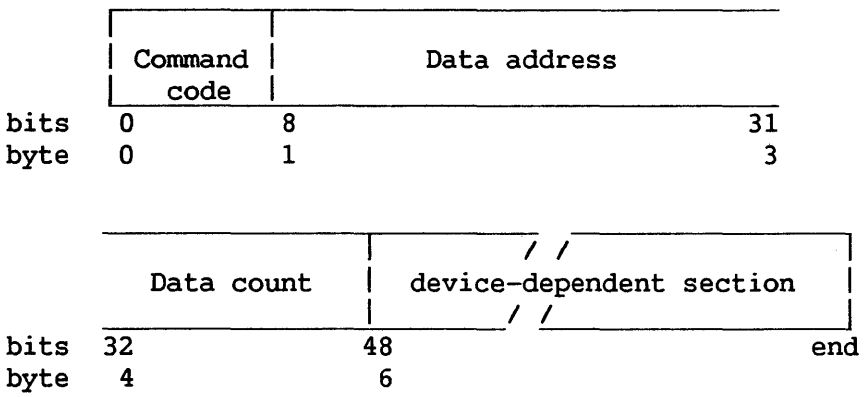


Figure 9-10. I/O Command Word (IOCW) Format

The fields in the IOCW are allocated as follows:

- Command code -- Bits 0-7 specify the operation to be performed.
- Data address -- Bits 8-31 specify a fullword-aligned physical memory address. This address is the beginning of the data area for the specified operation, or is the beginning of an indirect data address list, which in turn specifies the data areas for the operation.
- Data count field (DC) -- Bits 32-47 specify the number of 8-bit byte locations in memory to be transmitted either to or from the device. The data length may be up to 64K, minus 1.

0When the IOCW contains an invalid field, an I/O interrupt is generated with the invalid condition indicated in the IOSW.

9.8.1 Command Code

The command code, bit positions 0-7 of the IOCW, specifies to the I/O device the operation to be performed.

Bits 0 and 1 of the command code are the command type, and Bits 2-7 are the command modifier bits. The following four command types are defined:

```
Reserved - '00'
Read     - '01'
Write    - '10'
Control  - '11'
```

A Control command is used to initiate an operation that does not involve transfer of data, such as Seek or Write Tape Mark. For most control functions, the entire operation is specified by the modifier bits in the command code. If the command code does not specify the entire control function, the device-dependent field of the IOCW can be used. The data address field is always ignored for a control command.

9.8.2 Command Modifier Bits

The use of the modifier bits is device dependent. The modifier bits of the command specify to the device how the command is to be executed. The fifth modifier bit (Bit 6 of the command code) is set to indicate indirect data addressing for those devices supporting that option.

9.8.3 Definition of Storage Area

The IOCW defines a main memory area associated with an I/O operation by specifying the fullword-aligned address of the first byte to be transferred and the number of consecutive bytes contained in the area. The address of the first byte appears in the data-address field of the IOCW, unless indirect data addressing is specified. For indirect data addressing, the data address field of the IOCW addresses the beginning of the first entry of an indirect data address list. The number of bytes contained in the memory area is the data count (DC).

If the IOCW refers to a location not provided in the system, an I/O interrupt is generated with the "memory address error" condition indicated in the IOSW.

Programming Note

A malfunction that affects the validity of data transferred in an I/O operation is signaled at the end of the operation by the stored IOSW. In order to make use of the checking facilities provided in the system, data read in an input operation should not be used until the end of the operation has been reached and the validity of the data has been checked. Similarly, cn writing, the copy of data in main memory should not be destroyed until the program has verified that no malfunction affecting the transfer and recording of data was detected.

9.8.4 Indirect Address Lists

Certain devices expedite the transfer of multiple, noncontiguous pages of data to or from memory by means of an indirect address list, as indicated by a modifier bit of the command byte of the IOCW. The indirect address list is composed of 4-byte entries, each consisting of a fullword physical memory address. The IOCW data address field addresses the start of the indirect address list; the list in turn addresses the data areas for the operation. Data transfer begins into or from the first address specified and continues until a page (2 KB) boundary is reached. Data transfer then continues into or from the address specified in the second and succeeding list entries and continues for the length specified in the IOCW or until end-of-data at the device occurs. Theindirect address list thus allows the transfer of multiple, noncontiguous pages of data by means of a single IOCW.

Certain devices (especially disk devices) may require that the memory addresses specified in indirect address list entries have up to eleven low-order 0s (i.e., be aligned on a boundary as large as 2 KB). Refer to specific device descriptions in Chapters 10-13 for the restrictions applicable to particular devices.

9.8.5 Device-Dependent Section

The device-dependent section of the IOCW is not required by all devices. Its length depends on the type of device for which it is used. One use for this area is the sector address for a disk drive.

9.9 I/O STATUS WORD (IOSW)

All communication from I/O devices to the system occurs through IOSWs. An IOSW is stored at main memory Location X'00' when the associated I/O interrupt is granted. It is from one to eight bytes in length. The format of the IOSW is shown in Figure 9-11:

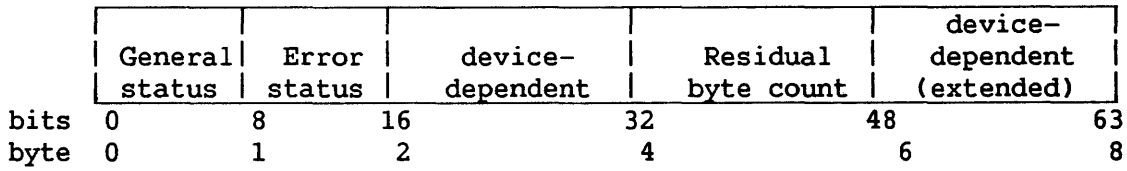


Figure 9-11. IOSW Format

Sections 9.9.1 through 9.9.4 provide an overview of the fields in the IOSW, which are discussed in more detail in Sections 9.10 through 9.12. One byte of the IOSW, the general status byte, is always stored. Additional bytes are stored as required by particular devices. A given type of device always stores an IOSW of the same length.

9.9.1 General Status Byte

The general status byte (Byte 0) is always stored.

Bits	Mnemonic	Meaning
0	IRQ	Intervention required
1	NC	Normal completion
2	EC	Error completion
3	U	Unsolicited
4	PC	IOP now ready (VS100 only)
5	DAR	Data area early release
6-7		Reserved

9.9.2 Error Status Byte

The error status byte (Byte 1) is always stored if the error completion bit is set in the general status byte. This byte may or may not have any error indications in it if the error completion bit is set. If any of the conditions listed in the error status bits occur, the corresponding flag is set and the error completion bit is set.

On VS300 systems, if Bit 9 (MPE) or Bit 10 (MAE) is set, IOSW Byte 7 further defines the error condition.

Bits	Mnemonic	Meaning
8	IC	Invalid command
9	MPE	Memory parity error
10	MAE	Memory address error
11	DM	Device malfunction
12	DAM	Memory or device damage (error after data transmission)
13	IL	Incorrect length
14-15	PP,DP	
	=11 (DCT)	A device configuration table is required by the IOP before any normal I/O operation can be performed on programmable devices.
	=10 (PP)	A peripheral processor microprogram is required by the IOP before any normal I/O operation can be performed on programmable devices.
	=01 (DP)	A device processor microprogram is required by the IOP before any normal I/O operation can be performed on programmable devices.

9.9.3 Device-Dependent Status Bytes

The device-dependent status bytes (Bytes 2-3) are different for each type of I/O device. Refer to specific descriptions of device types in Chapters 10-13 for further information on these bytes.

9.9.4 Residual Byte Count

The residual byte count indicates the byte count remaining at the time of I/O completion. Not all devices support storing of this count. If supported, this field is always stored when IL is set. If the device does not support storing of the residual byte count, it may still set the IL bit.

9.9.5 Extended Device-Dependent Status Bytes

On the VS300, Byte 7 is used to define errors indicated by Bit 9 (MPE) and Bit 10 (MAE).

9.10 GENERAL STATUS BYTE

9.10.1 IRQ -- Intervention Required

This bit is set with Error Completion (EC) and without normal completion (NC) to indicate that the device was in a not-ready state when an SIO or CIO instruction was accepted. This condition requires operator intervention to return the device to the ready state. IRQ is also indicated when the device becomes not ready during an I/O operation. In this case it always appears by itself (no other general status bit set), and completion is indicated later, when the "intervention required" condition has been cleared and the operation has been completed.

9.10.2 NC -- Normal Completion

This bit is set to indicate completion of an I/O operation without permanent error. An interruption with NC or EC set will occur exactly once for each SIO accepted.

9.10.3 EC -- Error Completion

This bit is set to indicate completion with error of an I/O operation. If NC is also set, the operation was successful after at least one retry by the device or I/O processor. If the EC bit is set, the errors detected are indicated in the error status byte or device-dependent status bytes, whether or not NC is also set. Possible combinations of NC and EC bit settings are listed below.

NC	EC	Meaning
0	0	Completion not indicated
1	0	Normal completion
0	1	Completion with permanent error
1	1	Completion with corrected error

9.10.4 U -- Unsolicited (Attention/Device Now Ready)

This bit is set when the device signals an unsolicited interrupt. An unsolicited interrupt is one not caused by I/O completion; it indicates that either the device has become available for I/O operations or that someone is signaling the CP for attention. This bit may be set with PC.

9.10.5 PC -- IOP Now Ready

This bit is significant only on the VS100. It indicates that an IOP may now accept an SIO, CIO, or HIO instruction. This bit can be set in an IOSW associated with any type of interrupt, with other bits, or by itself. Whenever an I/O machine instruction is rejected with condition code 2 (IOP BUSY), an interruption with PC set will eventually be presented. If more than one I/O machine instruction to devices on the same IOP is rejected with condition code 2 and no interruptions with PC set intervene between rejections, then only one interruption with PC set will be presented.

9.10.6 DAR -- Data Area Early Release

This bit indicates that the main memory page frame(s) containing a block of records written to a device will not be re-accessed and may be paged out by the operating system. No other status bits are set with DAR.

9.11 ERROR STATUS BYTE

9.11.1 IC -- Invalid Command

This indicates that part of the IOCW or the device-dependent control information was invalid (e.g., invalid command code or invalid data address alignment). This condition also causes a hard error to be indicated.

9.11.2 MPE -- Memory Parity Error

A memory parity error is indicated whenever there is a parity error while the I/O processor associated with the I/O device is accessing memory. On VS300 systems, the parity error is further defined in IOSW Byte 7.

9.11.3 MAE -- Memory Address Error

A memory address error is indicated whenever an attempt is made to an address outside the available memory on the machine during an I/O operation. On VS300 systems, the address error is further defined in IOSW Byte 7.

9.11.4 DM -- Device Malfunction

A device malfunction indicates that an equipment error has occurred during an I/O operation or that the I/O operation cannot be completed normally. A device malfunction is not indicated in the case where operator intervention will correct the problem. Therefore, device malfunction is not indicated when Intervention Required (IRQ) is set.

9.11.5 DAM -- Memory or Device Damage

This bit indicates that the data transfer was interrupted while in process and that data either at the device or in memory has been changed. The receiver of the data transmission has unpredictable data, and the data must be retransmitted (if possible) to correct the problem. The device's status may also have changed (e.g., for a magnetic tape, the tape may have been repositioned). DAM will be set only if the hard error indication is set.

9.11.6 IL -- Incorrect Length

This bit is set if the length of the data specified in the data count of the IOCW and the length of the corresponding item of data at the device are different. If this bit is set, the normal completion (NC) bit is also set. If the IL bit is set and the device supports storing of the residual data count, a valid residual data count will be stored. This count is zero when the IOCW data count is less than the number of bytes of data available at the device.

9.11.7 PP and DP -- IOP or Device Code Not Loaded

For IOCs, programmable IOPs, and programmable devices, these two bits are encoded to indicate that the required microprogram or table for the I/O operation is missing or is damaged.

Bit	(PP) 14	(DP) 15	Meaning
	1	1	A device configuration table is required by the IOP/IOC in order to process an I/O operation.
	1	0	Microprogram reloading is required for the IOP/IOC.
	0	1	Microprogram reloading for the programmable device is required in order to process an I/O operation.

9.12 EXTENDED DEVICE SPECIFIC STATUS BYTES (VS300 ONLY)

When the EC and MAE bits are on, IOSW Byte 7 shows one of the following hexadecimal codes:

Code	Meaning
02	Illegal system memory address
10	Illegal system memory page access
20	Illegal I/O command from IOC

When the EC and MPE bits are on, IOSW Byte 7 shows one of the following hexadecimal codes:

Code	Meaning
01	System memory data error
04	System bus memory read parity error
08	System bus parity error

9.13 I/O INSTRUCTIONS

SIO, CIO, and HIO are the privileged assembler instructions that control I/O operations. The SIO instruction starts a transfer of data between main memory and an I/O device via an I/O processor. The CIO instruction starts control operations for the I/O processor, or begins memory diagnostics or microcode loading or reading. The HIO instruction halts action started by a previous SIO or CIO. The format of all three instructions is as follows:

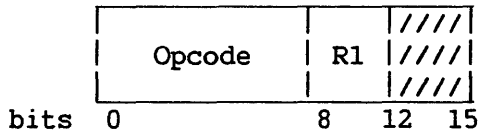


Figure 9-12. SIO, CIO, and HIO Instruction Format

The opcode is X'02' for SIO, X'0C' for CIO, and X'03' for HIO. The general register designated by R1 contains, in its 16 least-significant bits, the device address of the I/O device involved in the operation. Physical device addresses are described in Section 9.7.

9.14 INITIATION OF I/O OPERATIONS

The following sections describe the presentation of an I/O command by CP to I/O processor during execution of a SIO, CIO, or HIO instruction.

9.14.1 Initiation of I/O Operations -- VS15, VS65

When executing the I/O instruction, the CP locates the appropriate I/O Command Table (IOCT) by means of the Command Table Address (CTA) placed in the Device Adapter Status Table (DAST) prior to instruction execution. An I/O Command Word has been placed in the appropriate entry of the IOCT, also before instruction execution. The CP uses the physical device address, supplied by the SIO instruction, to index into these two tables.

The CP places in the BP-CP communications area the IOCW and command byte, which together define the requested I/O operation, and the PDA.

The CP checks the state of its a I/O request line to the BP and sets the condition code as follows:

Condition Code	Meaning
0	Command received by the BP
1	not used
2	not used
3	BP busy

9.14.2 Initiation of I/O Operations - VS100

When executing the I/O instruction, the CP locates the appropriate I/O Command Table (IOCT) by means of the Command Table Address (CTA) placed in the IOP Status Table (IOPST) prior to instruction execution. An I/O Command Word (IOCW) has been placed in the appropriate entry of the IOCT, also before instruction execution. The CP uses the physical device address, supplied by the I/O instruction, to index into these two tables.

The CP sends a 32-bit IPC message to the Bus Adapter (BA) specified in the PDA. The message identifies the IOP and device specified in the I/O instruction and defines the requested I/O operation as SIO, CIO, or HIO.

The BA maintains an IPC-IN register for each IOP. The BA ascertains whether the IPC-IN register for the specified IOP is available (that is, whether the IOP has processed the last message in its IPC-IN register). If the register is available, the BA loads it with the IPC message from the CP, signals the IOP, and waits for a response from the IOP.

The BA returns an IPC message to the CP, indicating whether the I/O command has been accepted or rejected. In the case of rejection, the message indicates two possible causes of rejection: the IPC-IN register for the IOP was unavailable, or the IOP was busy (that is, did not respond to the BA's signal).

After receipt of the BA's IPC message, the CP sets the condition code as follows:

Condition Code	Meaning
0	Command received by the IOP
1	not used
2	IOP busy
3	IPC-IN register busy

9.14.3 Initiation of I/O Operations -- VS300

When executing the I/O instruction, the CP locates the appropriate I/O Command Table (IOCT) by means of the Command Table Address (CTA) placed in the IOC Status Table (IOCST) prior to instruction execution. An I/O Command Word (IOCW) has been placed in the appropriate entry of the IOCT, also before instruction execution. The CP uses the physical device address, supplied by the I/O instruction, to index into these two tables.

The CP sends a 32-bit IPC message to the System Bus Interface (SBI). The message identifies the IOC and device specified in the I/O instruction and defines the requested I/O operation as SIO, CIO, or HIO.

The SBI determines whether the IPC register of the specified IOC is available (that is, whether the IOC has processed the last message in its IPC register). If the register is available, the SBI loads it with the IPC message from the CP.

The SBI sets a status bit to indicate that the IOC did or did not receive the IPC message. After inspecting the status bit, the CP sets the condition code as follows:

Condition Code	Meaning
0	Command received by the IOC
1	not used
2	not used
3	IOC busy or nonexistent

9.15 RECEIPT OF I/O COMMAND BY I/O PROCESSOR

When the I/O processor (BP, IOP, or IOC) receives an I/O command, it ascertains the status of the device addressed by the instruction.

There are four possible device states:

1. No previous command is active at the device and no interrupt signaled by this device is outstanding.

   A device in this state is considered available. The I/O processor accepts an SIO or CIO command for an available device. A HIO command is ignored; it is not acknowledged by an IOSW or SQB.

2. The device is busy with a prior command and has not yet signaled a completion interrupt.

   A device in this state is considered busy. The I/O processor rejects a SIO or CIO command; it sets the RDB (Rejected -- device busy) bit in the SQB that it writes after the device signals a completion interrupt.

   A HIO command, if successful, forces termination of the I/O operation in progress; the I/O processor stores a completion IOSW before raising its interrupt line. The SQB for this interrupt does not indicate any rejection.

3. The device has signaled an interrupt; the I/O processor has stored an IOSW for this interrupt and has raised its request line to the CP.

   A device in this state is considered available. The I/O processor accepts the SIO or CIO command. A HIO command is ignored; it is not acknowledged by an IOSW or SQB.

4. The device has signaled an unsolicited interrupt for which the I/O processor has not yet pre-stored an IOSW.

   The I/O processor rejects an SIO or CIO command; it sets the RIP (Rejected--interrupt pending) bit in the SQB that it writes when pre-storing an IOSW for this interrupt. An HIO command is ignored; it is not acknowledged by an IOSW or SQB.

9.16 I/O TERMINATION

An I/O operation lasts until one of the following events occurs:

The device completes the operation.
A HALT I/O instruction forces completion.
The device malfunctions.
The system is initialized or reset.

9.16.1 Completion

In the case of a data transfer operation, the device completes the operation when the number of bytes specified by the IOCW has been transferred.

9.16.2 Forced Completion

A HALT I/O instruction, if accepted and successful, immediately terminates any operation active at the device. Successful termination is reported by a normal completion IOSW with a nonzero residual byte count; an unsuccessful attempt is reported by an error completion IOSW. The HIO instruction does not clear any pending interrupts generated by the device.

9.16.3 Malfunction

When equipment malfunctioning is detected, the recovery procedure and the subsequent states of the devices depend on the type of error. Normally, the device attempts all appropriate error recovery procedures. If the recovery is successful, the I/O operation is completed and the IOSW indicates a corrected error. If the recovery is unsuccessful, the operation is terminated and a permanent error is indicated in the IOSW. (The representation of corrected and permanent errors is described in Section 9.10.3.)

9.16.4 System Initialization

All I/O devices are reset when a system IPL sequence is completed or when the LOAD button is pushed. Resetting causes I/O devices to terminate all operations. Status information and interrupt conditions in the devices are lost. Data transfer operations and control operations are immediately terminated, and the results are unpredictable.

9.17 I/O INTERRUPTIONS

9.17.1 Types of Interruption

I/O interruptions provide a means for the system to change its state in response to conditions that occur in I/O devices and I/O processors. A general distinction is made between a solicited and unsolicited I/O interrupt. A solicited interrupt results from completion of an I/O operation initiated by a SIO, CIO, or HIO instruction. An unsolicited interrupt results from a condition other than the completion of an I/O operation, such as operator action at an I/O device.

On the VS100, there exists a third type of interruption -- IOP NOW READY. An IOP NOW READY interruption occurs when an IOP becomes available for an SIO or CIO command after that command was previously rejected with a condition code of 2 (IOP BUSY).

9.17.2 Priority of Interrupts

All I/O interrupt requests to the CP are asynchronous with system activity; interrupt conditions associated with more than one I/O device may exist at the same time. Priority among I/O interrupt requests is determined by the physical position of the associated I/O processor in the hardware configuration, which is determined at installation time. While the processor is servicing one interrupt, the others remain pending.

9.18 INTERRUPT PROCESSING

9.18.1 Interrupt Processing -- VS15, VS65

At the time of a request for an I/O interrupt, the BP has already pre-stored the IOSW, SQB, and PDA in the appropriate locations (refer to Section 9.3) in main memory.

The CP grants an interrupt after ascertaining the DA and device number of its origin from the pre-stored PDA. It then copies the IOSW into Location X'00' of main memory, the SQB into Location X'50', and the PDA into Location X'80'. Finally, an I/O interrupt is formally granted by replacing the current PCW with the New I/O Interrupt PCW and storing the replaced PCW in the Old I/O Interrupt PCW location.

9.18.2 Interrupt Processing -- VS100

At the time of a request for an I/O interrupt, the IOP has already pre-stored the IOSW in the appropriate slot of the IOPST in main memory (refer to Section 9.3.) and has pre-stored the appropriate logical device number, along with the SQB, in the IPC-OUT register of the BA.

The CP grants an interrupt after ascertaining the BA number and IOP number of its origin from the Interrupt Request Mask (IRM) of the BA, an internal register that displays the pending I/O interrupt requests of the associated IOPs. It then copies the IOSW into Location X'00' of main memory, the SQB into Location X'50', and the PDA (formed by combining the BA number, IOP number, and logical device number of the device involved) into Location X'80'. Finally, an I/O interrupt is formally granted by replacing the current PCW with the New I/O Interrupt PCW and storing the replaced PCW in the Old I/O Interrupt PCW location.

9.18.3 Interrupt Processing -- VS300

At the time of a request for an I/O interrupt, the IOC has already pre-stored the IOSW, SQB, and PDA in the appropriate locations (refer to Section 9.3) in main memory.

The CP grants an interrupt after ascertaining the IOC of its origin by scanning the IOSW active bits in the IOCST. It then copies the IOSW into Location X'00' of main memory, the SQB into Location X'50', and the PDA into Location X'80'. Finally, an I/O interrupt is formally granted by replacing the current PCW with the New I/O Interrupt PCW and storing the replaced PCW in the Old I/O Interrupt PCW location.

CHAPTER 10
WANG WORKSTATION CHARACTERISTICS

10.1 INTRODUCTION

This chapter describes the control of data transfer between Wang VS systems and Wang VS serial workstations during data processing operations.

The VS workstation has two main parts, the CRT and the keyboard. The keyboard described in this chapter is the Wang Universal Keyboard. The description of keyboard functionality in Sections 10.2.5 through 10.2.7 and in Section 10.3 applies to the Wang model 4230 workstation equipped with the Wang Universal Keyboard.

10.2 THE CRT

10.2.1 Screen and Cursor

The CRT screen can display 24 rows of 80 characters each. Every position of the screen can display any character. A special symbol, resembling an underscore and called a cursor, is displayed beneath a character position to indicate where the next character entered from the keyboard will be stored. The cursor is displayed on the screen when data can be typed by the operator. If it is not displayed, the keyboard is locked. This has no effect on the display or the computer interface with the workstation, but prevents data entry from the keyboard. Each position of the screen is referenced by its row and column numbers. The first position of the screen (upper left corner) is called Row 1, Column 1. The columns are numbered from left to right and the rows from top to bottom. Position 2 is the second character from the left on the first line. Table 10-1 gives the set of characters displayed at the Wang model 4230 workstation and their associated representations in bits.

Table 10-1. The Character Set

NOTE: b_0 always equals zero*.	b_1 →	0	0	0	0	1	1	1	1
	b_2 →	0	0	1	1	0	0	1	1
	b_3 →	0	1	0	1	0	1	0	1

b_4	b_5	b_6	b_7	High-Order Digit → / Low-Order Digit ↓	0	1	2	3	4	5	6	7
0	0	0	0	0		â	SP	0	@	P	°	p
0	0	0	1	1	♦	ê	!	1	A	Q	a	q
0	0	1	0	2	▶	î	"	2	B	R	b	r
0	0	1	1	3	◀	ô	#	3	C	S	c	s
0	1	0	0	4	→	û	$	4	D	T	d	t
0	1	0	1	5	⌙	ä	%	5	E	U	e	u
0	1	1	0	6	|	ë	&	6	F	V	f	v
0	1	1	1	7	¨	ï	'	7	G	W	g	w
1	0	0	0	8	´	ö	(	8	H	X	h	x
1	0	0	1	9	`	ü	)	9	I	Y	i	y
1	0	1	0	A	^	à	*	:	J	Z	j	z
1	0	1	1	B	■	è	+	;	K	[	k	§
1	1	0	0	C	‼	ù	,	<	L	\	l	£
1	1	0	1	D	↕	Ä	−	=	M	]	m	é
1	1	1	0	E	β	Ö	.	>	N	↑	n	ç
1	1	1	1	F	¶	Ü	/	?	O	—	o	¢

*Bit combinations 10000000 through 11111111 are field attribute characters.

10.2.2 Screen Formatting

An important feature of the workstation screen is its division into fields. The beginnings of fields on the screen cannot generally be determined by inspection, except for high intensity fields, which are easily distinguished by their bright or blinking display. Although not visible, fields are very important, because they affect the operation of the workstation under both keyboard control and computer control. A field is defined as all characters from one field attribute character to the next.

A field can be from 0 to 80 characters in length. All the characters within a given field have the same attributes, which are defined by the field attribute character that precedes the field. The possible attributes are defined in Table 10-2.

Table 10-2. Field Attribute Character Values

Bit	Field Description
0	Must be 1
1	Selected-field tag for READ ALTERED and WRITE SELECTED
2	= 1 Underscore
3-4	Display control = 00 Intensified display = 01 Low intensity display = 10 Blinking display = 11 Nondisplay
5	Protect bit = 0 Modifiable field = 1 Protected field
6-7	Valid data specification = 00 Alphanumeric upper- and lowercase = 01 Alphanumeric uppercase shift = 10 Numeric only = 11 reserved

Field attribute characters are never displayed regardless of their value. Each row is considered to have a field attribute character just before the first character in the row and just after the last character in the row. These nondisplayed field attribute characters do not take up space on the screen. They have a default value of low intensity, protected, and alphanumeric upper- and lowercase. (Refer to the description of the field attributes in Section 10.2.3.) These default field attribute characters allow the use of 80-character lines. In addition, any location on the screen can contain a field attribute character.

10.2.3 Field Attributes

The meaning of each field attribute bit is given below.

Selected field tag: This field has been modified by user data entry at the workstation, or (when set in the mapping area) is written by Write Selected.

Underscore: The characters in this field are underscored when displayed on the screen.

Intensified display: The characters in this field are displayed in higher intensity than those in a low-intensity display field.

Low intensity display: The characters in this field are displayed on the screen.

Blinking display: The characters in this field are displayed alternately in intensified display and normal display mode. The display will change modes at a fixed rate of about three times a second.

Nondisplay: The characters in this field are not displayed on the screen. The field will be displayed as all blanks.

Unprotected (also called "modifiable"): Any or all of the positions of this field can be changed by the operator.

Protected: No position of this field can be modified by the operator.

Alphanumeric: This field allows typing of any character on the keyboard.

Uppercase shift: Letters are displayed and stored as uppercase only, regardless of whether the SHIFT or LOCK key is pressed. All other keys respond to the SHIFT and LOCK keys as they normally would.

Numeric only: Only the characters 0-9, decimal point (.), and minus (-) may be entered into this field. If other keys are pressed, the keystroke is ignored and the alarm sounds.

Reserved: This is not a valid combination at this time. It is intended for addition of later options. Its use produces unpredictable results.

10.2.4 Tabs

Ten tabs can be set by programs; they can be set to any column of the workstation screen (1-80) with the command Write Tabs. They do not take up a screen location, are not displayed, and allow forward tabbing operations to stop at locations within modifiable fields. A tab position is specified by column number and affects the column of every row in which the specified column is modifiable. A tab set in a protected field has no effect; i.e., the cursor cannot be positioned to such a tab by either the TAB or BACKTAB key. When the workstation is powered on, all tabs are cleared.

10.2.5 Audio Indicators

The audible alarm sounds a short tone whenever an illegal keying operation is attempted. This operation can be an attempt by the user to enter data into a protected field, to move the cursor past the end of the screen with a field-sensitive key, or to enter data when the keyboard is locked. The alarm also sounds when a Write is issued while Bit 2 of the WCC is on.

The keystroke indicator is a small device attached to the keyboard that makes a clicking sound. It sounds each time a key is pressed.

The operator can adjust the volume of auto indicators by the following procedure:

1. Press the 2ND key and then PF5 to enter set-up mode. In set-up mode, bell, clicker, and arrow symbols appear at the bottom of the workstation screen. (The arrow symbol signifies the type-ahead feature, described in the next section.)

2. Position the cursor over the bell or clicker symbol, using the east/west cursor keys.

3. Set the volume using the north/south cursor keys. Pressing these keys causes the bell or clicker to sound. Each stroke of the north key increases volume until the maximum level is reached; each stroke of the south key decreases volume until the minimum level is reached.

4. Press PF16 to exit from set-up mode.

10.2.6 Type-Ahead

The type-ahead feature allows the operator to enter data from the keyboard while data is being transferred between the workstation and the VS. Using this feature, the operator need not wait for the next screen to appear before entering data to that screen.

The operator enables or disables the type-ahead feature by the following procedure:

1. Press the 2ND key and then PF5 to enter set-up mode. In set-up mode, bell, clicker, and arrow symbols appear at the bottom of the workstation screen.

2. Position the cursor over the arrow symbol, using the east/west cursor keys.

3. Press the up arrow to enable type-ahead; press the down arrow to disable the feature. When the feature is enabled, the arrow symbol blinks; when the feature is disabled, the arrow symbol is non-blinking.

4. Press PF16 to exit from set-up mode.

10.3 THE KEYBOARD

The VS workstation keyboard is illustrated in Figure 10-1.

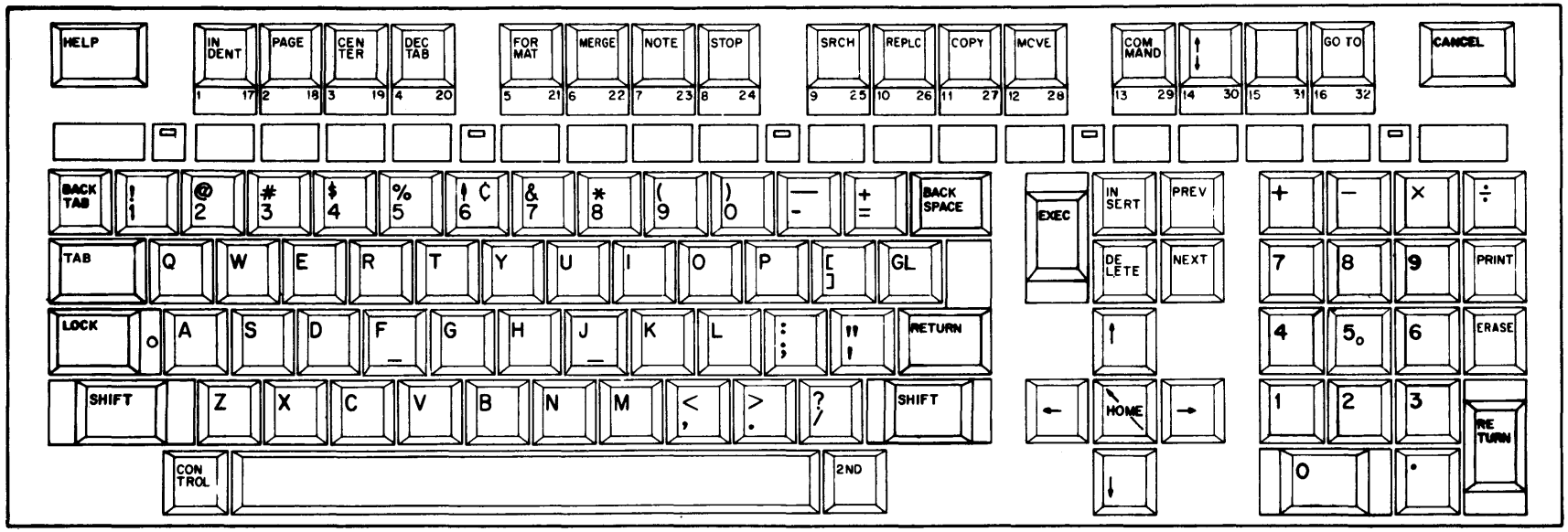


Figure 10-1. The Keyboard

10.3.1 Cursor Positioning Keys

Non-Field-Sensitive Keys

These keys position the cursor but are not affected in any way by fields and field attribute characters. They can position the cursor to any location on the screen. There are four keys in this group:

Up arrow	Positions the cursor in the same column but up one row. If the cursor started in the first row, it is positioned in the same column but in the last row.
Down arrow	Positions the cursor in the same column but in the next row. If the cursor started in the bottom row of the screen, it is positioned in the same column but on the first row of the screen.
Left arrow	Moves the cursor one position to the left in a row. If the cursor was at the start of a row, it moves the cursor to the last position in the preceding line. If the cursor is in the first location of the screen, it is positioned in the last position of the screen.
Right arrow	Moves the cursor one position to the right in a row. If the cursor is at the end of a row, it is moved to the first position of the next row. If it is at the last position of the screen, it is positioned in the first position of the screen.

Field-Sensitive Cursor Positioning Keys

The following keys move the cursor to the first byte of a modifiable field or to the first byte of a field with the attribute of protected numeric. (The contents of a field with this attribute may be alphanumeric; the attribute and contents of a protected field need not match.) Because of their sensitivity to field attributes, these keys can be used to simplify data entry. A tab set in a protected field has no effect on these keys.

TAB	Moves the cursor ahead to the start of the next modifiable or protected numeric field. If there are no more such fields, the alarm sounds and the cursor does not move.
BACK TAB	Positions the cursor at the start of the nearest modifiable or protected numeric field preceding the current cursor location. If the cursor is in a modifiable field and in other than the first location, the cursor is positioned to the start of that field. If there is no preceding modifiable location (or protected numeric field), the alarm sounds and the cursor does not move.

EXECUTE — Advances the cursor to the first position of the next line, and then moves the cursor to the first modifiable or protected numeric field following the start of the line. This key may cause the cursor to be moved several lines from the original position. If there is no modifiable or protected numeric field following the start of the next line, the alarm sounds and the cursor does not move.

HOME — Positions the cursor at the first modifiable or protected numeric field on the screen. If there are no such fields on the screen, the alarm sounds and the cursor does not move.

10.3.2 Data Entry Keys

None of the keys mentioned so far change data in any positions of the screen display. The sole function of the data entry keys is to enter data into positions of the screen. For all these keys the cursor must be in a modifiable field. If the cursor is not in a modifiable field, the keystroke is not honored and the alarm sounds.

Character keys — These include letters, numbers, and special characters. These keys enter characters just as a typewriter does (with the use of LOCK and SHIFT). If any characters other than numerals (0-9), hyphen (-), or period (.) are pressed in a numeric attribute field, an audible alarm sounds. If the field is an uppercase character attribute field, lowercase letters are interpreted as uppercase letters.

When the cursor is in the last position of a field and one of these keys is pressed, the character is entered into the location and the cursor is positioned at the next modifiable location. This may involve skipping the field attribute character or skipping several lines. If the cursor is currently at the last modifiable location on the screen, the keystroke is honored, the alarm sounds, and the cursor is not moved.

ERASE — Sets the cursor location and all subsequent locations of the current field to blank characters. Any locations that precede the cursor are not changed. The cursor does not move.

INSERT — Places a blank at the cursor location and shifts to the right by one position all the characters in the current field, starting with the one at the cursor location up to but not including the last character in the field. The last character in the field, if a blank, pseudoblank, or dectab, is lost. If the last character in the field is not a blank, pseudoblank, or dectab, no screen location is changed, the alarm

sounds, and the cursor does not move. Pseudoblanks and dectabs are the characters X'0B' and X'05', respectively, in a modifiable field.

DELETE — Deletes the character at the cursor location and moves the subsequent characters in the field left by one position. The last character moved is the rightmost character in the field, and it is followed by a newly inserted blank. If the cursor is not in a modifiable field, the key is not honored and the alarm sounds. This key is reciprocal in action to the INSERT key.

10.3.3 Special Keys

Special keys are keys additional to those already described that are used during data entry.

SHIFT — Has the same effect as the SHIFT key on a typewriter. For keys with an upper and lower character on the key face, the SHIFT key is used to select which character is to be entered. However, it has no effect on letters to be entered in an uppercase attribute field. These are entered as uppercase whether the SHIFT key is pressed or not. Pressing this key when the SHIFT light is lit causes the SHIFT light to be turned off and unSHIFTs the keyboard.

LOCK — Lights the SHIFT light. The workstation then behaves as if the SHIFT key were continuously pressed. Pressing this key again does not change the device status. Pressing the SHIFT key turns off the SHIFT light, returning the keyboard to an unSHIFTed state. When the workstation is powered on, the device is in an unLOCKed state.

2ND — Used to adjust the volume of audio indicators and to enable/disable the type-ahead feature. This key is used in conjunction with PF5, PF16, and the cursor control keys, as described in Sections 10.2.5 and 10.2.6.

CANCEL — Causes all field attribute characters on the screen with a blinking display to be set to (unblinking) high intensity. This key is still effective when the keyboard is locked for data entry.

10.3.4 Keys that Communicate with the Computer

This set of keys causes an interruption to be presented to the computer. If the key can be honored, the AID byte in the IOSW is set to the character for the struck key and an interruption is presented to the computer. After these keys are pressed, all keys except the HELP key and the CANCEL key are locked, and the alarm will sound if they are struck. The cursor is removed from the screen.

HELP	This key is intended for operating system use. The SHIFT key does not affect its action. The key cannot be honored when an unsolicited interruption is pending for the same device or when the key has been disabled by a system administrator via SECURITY. At any other time the key is honored, both when the keyboard is locked for any of the data entry keys and during a Read or Write to the workstation. A HELP key struck while a Read or Write is in progress results in a separate attention interruption occurring after the Read or Write completion interruption.
PF1-PF32	Program function keys -- There are 16 PF keys; the lowercase values for these keys represent PF1-PF16, and the shifted (uppercase) values PF17-PF32. These keys work the same as RETURN, the only difference being in the value of the AID byte generated.
RETURN	This key is the normal means of terminating data entry and requesting the program to process the data. The SHIFT key does not affect the action of the RETURN key. The RETURN key, like a PF key, is not honored when the keyboard is locked for data entry keys.

10.4 DATA AREA

I/O commands transfer data between the workstation and an area of main memory referred to as the data area, whose beginning and length are specified by the IOCW.

The data area can consist of two adjacent areas: the order area and, optionally, the mapping area. The 4-byte order area contains the starting row number of the Read or Write operation to be performed. On a Write, the Write control character (WCC) of the order area specifies how the Write is to be performed. After a Read, the order area receives the current column and row address of the cursor. The mapping area is the data transmitted to or from the screen and contains field attribute characters and display characters. Figure 10-2 illustrates the relationship of data area, order area, and mapping area.

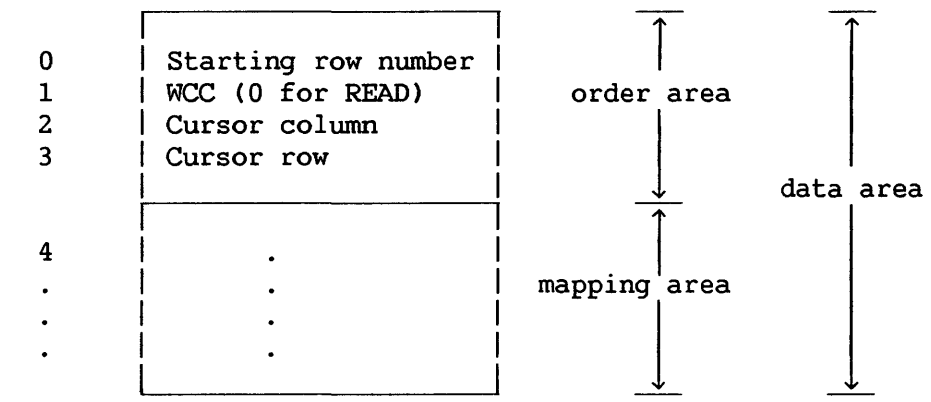


Figure 10-2. Data Area Specified by Workstation IOCW

10.4.1 Order Area

The order area is always four bytes long. Table 10-3 shows the layout of this area.

Table 10-3. Significance of Bytes in the Workstation Order Area

Byte	On Read	On Write
0	Row number	Row number
1	Reserved (must be 0)	WCC (write control character; Refer to Sections 10.4.3 and 10.4.4)
2	Cursor column address	Cursor column address (if cursor bit set in WCC)
3	Cursor row address	Cursor row address (if cursor bit set in WCC)

The contents of the order area and the interpretation of the fields in the area are different for a Read and a Write.

10.4.2 Interpretation of the Order Area on a Read

The first byte of the order area is inspected before the data transfer and is used to specify the starting row number for the Read. If this row number is not in the range 1-24, the command is terminated with an indication of Order Check (OR) in the IOSW. This byte is not changed by the Read.

The third and fourth bytes of the order area are set by the Read to the address of the cursor at the time of the read. The first byte of the two will contain the column number (1-80), and the second will contain the row number (1-24) of the current cursor location. These two bytes are not inspected before the Read.

The second byte of the order area for a Read is not inspected or modified, but is to be supplied as binary 0s for compatibility with future options.

10.4.3 Interpretation of the Order Area on a Write

Neither the order area nor the mapping area is changed on a Write. The first byte of the order area on a Write is interpreted as the row number where the Write is to start. If this row number is not in the range 1-24, the command is terminated with an indication of order check (OR) in the IOSW.

The second byte of the order area is interpreted as the Write Control Character (WCC). If the "position cursor" bit is set in the WCC, the next byte of the order area is interpreted as a cursor column address, and the fourth byte as the cursor row address. If the "position cursor" bit is not set in the WCC, the third and fourth bytes of the order area are ignored.

If the "position cursor" bit is set in the WCC, the cursor row address byte must be set to a value between 0 and 24 inclusive, and the cursor column address byte must be set to a value between 0 and 80 inclusive. After the Write completes, the cursor is positioned to that row and column. If the cursor row address byte is 0, it is treated as if it were 1. If the cursor column address byte is 0, this acts as if the cursor were positioned one location before the first location in the specified row and the TAB key were pressed. If there are no modifiable positions on the screen after the Write command, the cursor is positioned to the first location in the specified row.

If the "cursor position" bit is set in the WCC and the cursor row address byte has a value other than 0-24 or the cursor column address byte has a value other than 0-80, the command is terminated with an indication of Order Check (OR) stored in the IOSW.

10.4.4 Write Control Character (WCC)

The Write Control Character (WCC) is is the second byte of every order area supplied for a Write operation. WCC bits can be set to select the options shown in Table 10-4.

Table 10-4. Workstation Write Control Character (WCC) Codes

Bit	Explanation (if set to 1)
0	Unlock keyboard (Lock if 0)
1	Sound alarm
2	Position cursor
3	Roll down
4	Roll up
5	Erase modifiable fields to pseudoblanks
6	Erase and protect rest of screen
7	Reserved (must be 0)

WCC options, if selected, and data transfer occur in the following order:

1. The keyboard is locked.
2. The alarm is sounded.
3. The roll down is performed.
4. The roll up is performed.
5. The "erase modifiable" or "erase and protect rest" is performed.
6. The data is transferred to the screen.
7. The keyboard is unlocked.
8. The cursor is positioned.
9. If the keyboard is unlocked, the cursor is displayed.

Unlock the Keyboard

After the record is written to the screen and after sounding of the alarm, if specified, the AID character is set to blank, and the keyboard is then unlocked.

If bit 0 is 0, the keyboard is locked before any data is transmitted to the workstation. If the keyboard is locked, this bit does not change the status of the keyboard. The normal method for locking the keyboard is to wait for the operator to press one of the computer communication keys. If the bit is 0 and the keyboard is locked, the AID character in the IOSW does not change. However, if the command locks the keyboard, the AID character is set to " ' " (X'21').

Sound the Alarm

If Bit 1 is set to 1, the alarm sounds before the data is transmitted to the screen.

Position the Cursor

If Bit 2 is set to 1, after data is transferred to the screen the cursor is positioned as described in Section 10.4.3.

Roll Down

Setting Bit 3 to 1 causes the bottom line of the screen to be lost and each line above it to be copied into the next lower line. This copying proceeds until the row specified in the order area has been copied. The specified row is then set to blanks and the Write continues.

Roll Up

If Bit 4 is set to 1, the row specified in the order area is lost and each line below it is copied into the next higher line (e.g., line 1 is replaced by the contents of line 2, etc.). This copying proceeds until the last row of the screen has been copied. The last row is then set to blanks, and the Write proceeds on the last line of the screen. An attempt to write more than one line in a single command with "roll up" specified results in Order Check (OR) being reported.

Erase Modifiable Fields to Pseudoblanks

All modifiable locations at and after the row address specified in the order area are set to pseudoblank characters (bit pattern 00001011) before the data is transferred to the screen.

Erase and Protect Rest of Screen

All locations of the screen at and after the row address specified in the order area are set to the field attribute character X'8C' before the data is transferred to the screen. Therefore, there are no modifiable locations after the data that is written.

10.4.5 Mapping Area

The mapping area contains the data transmitted either to or from the screen. Its maximum length is 1920 bytes. The first location of the mapping area corresponds to the first character of the row specified in the first byte on the order area. Byte number 81 of the mapping area would correspond to the first byte of the next row. If the starting row number and the length of the mapping area are such that locations in the mapping area would extend past the end of the screen, the command will be terminated with an indication of incorrect length stored in the IOSW. Note that, although the mapping area's first position always corresponds to the start of a row, the only restriction on the end of the mapping area is that it not extend past screen end. No mapping area need be supplied for a 4-byte Read or Write (order area only).

10.5 WORKSTATION IOCW

A general discussion of the IOCW is found in Chapter 9. The workstation IOCW consists of a command, a data area address, and data count, as shown in Figure 10-3.

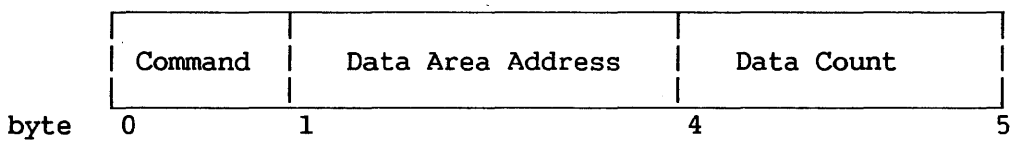


Figure 10-3. Workstation IOCW

10.5.1 Command Byte

The first byte of the IOCW contains the I/O command, the command modifier bits, and the indirect addressing flag in the form:

CCMMMMI0

where CC is the command, MMMM are the command modifier bits, and I is the indirect data addressing flag. The last bit of the command byte is always 0.

Table 10-5 lists valid workstation commands.

Table 10-5. Workstation Commands

Command	Command and Modifier Bits	
WRITE	10	0000
WRITE SELECTED	10	0100
WRITE TABS	10	0001
READ	01	0000
READ ALTERED	01	0100
READ DIAGNOSTIC	01	0010
READ TABS	01	0001

Indirect Addressing

Bit 7 of the command byte is set to indicate that the data address portion of the IOCW addresses an Indirect Address list as described in Chapter 9. For the workstation, the address contained in the first entry of an Indirect Address list must have two low-order 0s (specifying word alignment), and the Indirect Address list itself must be word aligned.

Data Address

The data address points to the first byte of the order area, or to an Indirect Address list whose first entry points to the first byte of the order area. As explained in Section 10.4, the first byte of the order area is also the first byte of the data area.

Data Count

The data count specifies the number of bytes in the data area, which as explained in Section 10.4, consists of the order area and mapping area. The minimum data count permitted for this device is the length of the order area (four bytes). If a length shorter than the order area is specified, the command is terminated with an indication in the IOSW of incorrect length. The mapping area must not extend past the end of the workstation screen, so the maximum length of the mapping area is 1920 bytes and the maximum data count is 1924 (mapping area plus order area). If the data count exceeds 1924, the command is terminated with an indication of incorrect length stored in the IOSW.

10.6 WORKSTATION I/O COMMANDS

10.6.1 Read Command

The Read command causes the contents of the screen locations that correspond to the mapping area to be copied into the mapping area. This includes all characters and field attribute characters in the range to be read. Selected-field tags of the field attribute characters in the portion of the screen read are turned off, both in the workstation and in main memory. The cursor row and column addresses are stored in the order area. This command is valid both when the keyboard is locked and when it is unlocked. Issuing it is not recommended while the keyboard is unlocked, however, as this may cause some operator keystrokes to be lost.

If any of the characters in the range of the Read are pseudoblanks, they are converted to blanks on the screen before the data is read. Pseudoblanks are characters with bit patterns 00001011 (the solid blank) or 00000101 (the dectab) in a modifiable field. When these characters are in protected fields, they are not considered to be pseudoblanks, and they are not changed to blanks either on the screen or in memory.

If any characters in the range of the Read are field attribute characters with blink indicated, the blink indication is converted to high intensity both on the screen and in memory. This is true for field attribute characters that have either protected or modifiable field indications set.

10.6.2 Read Altered Command

The Read Altered command causes the contents of fields within the specified range that have selected-field tags set to be copied into corresponding positions of the mapping area. The selected-field tags in the portion of the screen read are turned off at the workstation, but they are set in the corresponding field attribute characters of the mapping area. Pseudoblanks and blinking fields within the range of the Read Altered are affected as for the Read command.

10.6.3 Read Diagnostic Command

The Read Diagnostic command is identical to the Read command, except that it does not change pseudoblanks to blanks, reset blinking fields, or turn off selected-field tags either on the screen or in the data that is read.

10.6.4 Read Tabs Command

The Read Tabs command reads into memory the column numbers of all set tabs. The command transmits up to ten characters. Each location has a value of 0 or 1-80. The first 0 encountered indicates that there are no more set tabs and that subsequent locations have undefined values. The tabs are listed in the mapping area in order of increasing column numbers. The IOCW must have a data count greater than or equal to 14 or the command is rejected with an indication of OR (order check).

10.6.5 Write Command

The Write command causes a transfer to the screen of the data in the mapping area. Field attribute characters, including selected-field tags, are transferred unchanged. This command can be issued when the keyboard is locked or unlocked. It is, however, normally undesirable to issue a Write when the keyboard is unlocked, because doing so could cause loss of operator keystrokes.

10.6.6 Write Selected Command

The Write Selected command causes a transfer to the screen of those fields in the mapping area that have selected-field tags set in their field attribute characters. The selected-field tags in main memory are not reset. Selected-field tags at the workstation (indicating altered fields) are turned off only in those field attribute characters that identify the fields to be written.

10.6.7 Write Tabs Command

The Write Tabs command causes all tabs to be cleared, and then sets up to 10 tabs specified in the first 10 bytes of the mapping area. Each column that is to be set as a tab stop has its column number specified in the mapping area. Column numbers are to be specified in increasing order (1-80). The first zero byte encountered within the 10-byte mapping area terminates the list of tab settings; the contents of any subsequent bytes are not examined. Incorrect specification of tab settings result in unpredictable and erroneous tab operation. The IOCW must have a data count greater than or equal to 14, or the command is rejected with an indication of OR (order check).

10.7 WORKSTATION I/O STATUS WORD

For a general discussion of the IOSW, refer to Chapter 9. Figure 10-4 shows the IOSW format for workstations. Byte 7 is used on the VS300 system only.

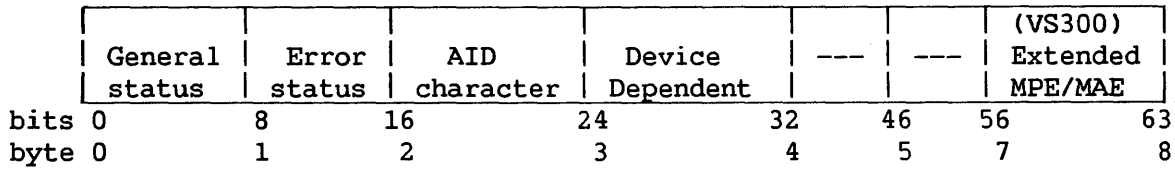


Figure 10-4. Workstation IOSW Format

10.7.1 General Status Byte

IOSW Bit	Mnemonic	Meaning
0	IRQ	Never set.
1	NC	Set on normal completion.
2	EC	Set on completion with error.
3	U	Set on power-on or on pressing of RETURN, PROGRAM FUNCTION, or HELP key. NC and EC never set along with this bit.
4	PC	Set only in conjunction with NC, EC, or U.
5-6		Reserved (always 0).
7		Reserved for software use.

10.7.2 Error Status Byte

IOSW Bit	Mnemonic	Meaning
8	IC	Set to indicate invalid command byte in IOCW, IOCW not fullword aligned, Indirect Address list not fullword aligned, or data area not fullword aligned. EC always set when IC is set.
9	MPE	Set with EC on occurrence of main memory parity error during reading of CTA, Indirect Address list, IOCW, or data. On VS300 only, is qualified by IOSW Byte 7.
10	MAE	Set with EC on occurrence of main memory addressing error during reading of IOCW, Indirect Address list, or data. On VS300 only, is qualified by IOSW Byte 7.
11	DM	Set on timeout during reading or writing of screen buffer in response to an I/O command. EC always set when DM is set.
12	DAM	Set when device RAM parity error or power-off occurs in an I/O operation.
13	IL	Set if IOCW specifies a data length less than 4, or if an operation attempts to read or write beyond the end of the screen. (In the latter case, the operation is terminated rather than suppressed.) EC always set when IL is set.

Bit	Mnemonic	Meaning
14	PP	Set when microprogram loading for a programmable IOP (data link processor) is required.
15	DP	Set when a programmable workstation is powered off, or when an internal RAM parity error occurs. This error condition indicates that device processor microprogram reloading is required. If both PP and DP are set, loading of a device configuration table for a programmable I/O processor is required.

10.7.3 Device-Dependent Bits

Bits 16-33 (Bytes 2-3) are always stored on an interrupt. Bits 16-33 hold the current Attention ID (AID) character. Table 10-6 lists AID characters and the key or condition associated with them.

IOSW Bit	Mnemonic	Meaning
16-23		The current AID character. This byte indicates whether the keyboard was locked by the last completed I/O operation, or it indicates what PF key was last struck. The AID character is X'20' if the operation is a Write which has unlocked a previously locked keyboard, X'21' if the operation is a Write which has locked a previously unlocked keyboard. If the operation did not change the locked/unlocked status, the AID character is the last AID character set by workstation interaction (computer communication key, power-on, or error).
24	OR	Order check. This indicates that the row or column addresses specified in the order area are invalid or that the IOCW data count was less than 14 for Write Tabs. The row specified in the order area is not between 0 and 24, or the column is not from 0 to 80, or more than 10 tabs were requested. The screen or tab settings may have been modified.
25		Set on normal completion of a Read Altered command that transfers data to main memory.
26-31		Reserved for future use.

Table 10-6. Attention ID (AID) Characters

AID	Hex Character (ASCII)	Graphic Character	AID	Hex Character (ASCII)	Graphic Character
Keyboard Unlocked	20	' ' (blank)	Locked by Write	21	'
RETURN key	40	@			
PF1 key	41	A	PF17 key	61	a
PF2 key	42	B	PF18 key	62	b
PF3 key	43	C	PF19 key	63	c
PF4 key	44	D	PF20 key	64	d
PF5 key	45	E	PF21 key	65	e
PF6 key	46	F	PF22 key	66	f
PF7 key	47	G	PF23 key	67	g
PF8 key	48	H	PF24 key	68	h
PF9 key	49	I	PF25 key	69	i
PF10 key	4A	J	PF26 key	6A	j
PF11 key	4B	K	PF27 key	6B	k
PF12 key	4C	L	PF28 key	6C	l
PF13 key	4D	M	PF29 key	6D	m
PF14 key	4E	N	PF30 key	6E	n
PF15 key	4F	O	PF31 key	6F	o
PF16 key	50	P	PF32 key	70	p
HELP key	30	0	Screen damage alert	3F	?

Additional Workstation AIDs

The following AID characters, in addition to those listed in Table 10-6, can be returned in the IOSW.

AID Character	Meaning
X'00'	Power On
X'01'	Disconnect
X'02'	Connect

10.7.4 Extended MPE/MAE Byte (VS300 Only)

On the VS300 only, IOSW Byte 7 provides extended status information on memory address and memory parity errors.

When the EC and MPE bits are both set to 1, IOSW byte 7 shows one of the following hexadecimal codes:

Code	Meaning
01	System memory data error
04	System bus memory read parity error
08	System bus parity error

When the EC and MAE bits are both set to 1, IOSW byte 7 shows one of the following hexadecimal codes:

Code	Meaning
02	Illegal system memory address
10	Illegal system memory page access
20	Illegal I/O command from IOC

10.8 EXAMPLE OF COMPUTER CONVERSATION WITH A WORKSTATION

When the operator powers on the workstation, an attention interrupt is generated, and the workstation microcode is loaded. The system then issues a Write. The data transmitted to the workstation formats the screen into fields and displays the information that tells the operator what data to insert into the fields. This Write has the WCC set to unlock the keyboard after the Write. After this Write is finished, the computer does not need to communicate with the workstation until the operator has signaled that data entry is finished and the system may read the data. Having finished entering data, the operator presses the RETURN key (or one of the other communication keys). This causes an interrupt and locks the data entry keys, program function keys, and RETURN key. At any time after this interrupt the program can issue a Read to the workstation. After the Read has finished, the program processes the data read and prepares new messages and a new screen format, which are sent to the workstation with a Write. The WCC has the bit set to unlock the keyboard. The above sequence of operations can be repeated until all needed data has been supplied to both the operator and the computer.

CHAPTER 11
WANG PRINTER CHARACTERISTICS

11.1 OVERVIEW

Five basic types of printers are supported by Wang VS systems:

- Chain Train
- Band
- Dot Matrix
- Daisy Wheel
- Laser

Characteristics of the various printer models are shown in Tables 11-1 through 11-6.

Transfer of data between main memory and the printer is facilitated by the IOCW (sent to the printer) and the IOSW (returned by the printer). The format of these structures and their use in communication between CP and peripheral devices is generally described in Chapter 9. Sections 11.7 and 11.9, respectively, describe the printer IOCW and IOSW in detail. Sections 11.2 through 11.6 provide an overview of the data blocks addressed by IOCWs, and of the IOCW/IOSW protocol.

Table 11-1. Characteristics of VS Chaintrain Printers

Wang VS Model Number	5570	5571
Printing Speed (lpm)	600	425
Size of Character Set	64 (Uppercase)	96 (Upper and Lowercase)
Chars/inch (Horizontal Pitch)	10	10
Maximum Characters per Line	132	132
Lines/Inch (Vertical Pitch)	6,8[a]	6,8[a]
Vertical Format Unit	Paper Tape Channels 1 - 12	Paper Tape Channels 1 - 12
Expanded Characters	No	No
Loadable Fonts	No	No

[a] 8 lines/inch is a hardware-selected option

Table 11-2. Characteristics of VS Band Printers.

Wang VS Model Number	5573	5574	5574-1	5575
Printing Speed (lpm)	250	600	600	1100
Size of Character Set	64 (Uppercase)	64 (Uppercase)	64/96	64/96
Characters per Inch	10/15	10	10	10
Maximum Characters per Line	132/10 pitch 198/15 pitch	132	132	136
Lines/Inch	6,8 (software selectable)	6,8 (software selectable)	6,8 (software selectable)	6,8 (software selectable)
Vertical Format Unit	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12
Expanded Characters	No	No	No	No
Loadable Fonts	No	No	No	No

Table 11-3. Characteristics of VS Daisy Wheel Printers

Wang VS Model Number	6581W	6581WC Wide Carriage	DW/20	DW/55
Printing Speed (cps)	30	30	20	55
Size of Character Set	96	96	96	96
Characters per Inch	10,12,15 (software selectable)	10,12,15 (software selectable)	10,12,15 (software selectable)	10,12,15 (software selectable)
Maximum Characters per Line	132/10pitch 158/12pitch 198/15pitch	180/10pitch 216/12pitch 270/15pitch	132/10pitch 158/12pitch 198/15pitch	132/10pitch 158/12pitch 198/15pitch
Lines/Inch	3,4,6,8 (software selectable)	3,4,6,8 (software selectable)	3,4,6,8 (software selectable)	3,4,6,8 (software selectable)
Vertical Format Unit	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12
Expanded Characters	No	No	No	No
Loadable Fonts	Yes (Each print wheel is a font)	Yes (Each print wheel is a font)	Yes (Each print wheel is a font)	Yes (Each print wheel is a font)

Table 11-4. Characteristics of VS Matrix Printers

Wang VS Model Number	5521	5531-2	5521I	5533-1/ 5535-1
Printing Speed (cps)	200	120	120	5533/100 5535/180
Size of Character Set	96 (Upper- and Lowercase)	96 (Upper- and Lowercase)	96 in normal mode	114
Chars/Inch (Horizontal Pitch)	10 (normal) 5 (expanded characters)	12 (normal) 6 (expanded characters)	10 (normal) 5 (expand.)	10 (5535-1) 12 (5533-1)
Maximum Characters per Line	132	132	128 (normal) 56 (ideo-graphic)	132/10pitch 158/12pitch
Lines/Inch	6	6	Approx: 8 (normal) 4 (graphic)	6/8
Vertical Format Unit	Paper Tape Channels 1 - 5	Paper Tape Channels 1 - 5	Paper Tape Channels 1 - 5	DAVFU Channels 1 - 12
Expanded Characters	Yes	Yes	Yes	Yes
Loadable Fonts	No	No	No	No
Ideographic Printing/ Graphics	No	No	Ideographic	No

(continued)

Table 11-4. Characteristics of VS Matrix Printers (continued)

Wang VS Model Number	5577
Printing Speed (cps)	160 @10-pitch draft 40 @10-pitch HD
Size of Character Set	128
Chars/Inch (Horizontal Pitch)	10, 12, 15 (font-dependent)
Maximum Characters Per Line	132/10pitch 158/12pitch 198/15pitch
Lines/Inch	3,4,6,8 (software selectable)
Vertical Format Unit	DAVFU Channels 1 - 12
Expanded Characters	No
Loadable Fonts	Yes
Ideographic Printing/ Graphics	Graphics

Table 11-5. Characteristics of VS Laser Printers

Wang VS Model Number	LPS-8	LIS-12	LIS-24
Printing Speed (pg/min)	8	12	24
Size of Character Set	128 (Upper- and Lowercase)	128 (Upper- and Lowercase)	128 (Upper- and Lowercase)
Chars/Inch (Horizontal Pitch)	10,12,15 (font-dependent)	10,12,15 (font-dependent)	10,12,15 (font-dependent)
Maximum Characters per Line	136/10pitch 163/12pitch 203/15pitch	136/10pitch 163/12pitch 203/15pitch	136/10pitch 163/12pitch 203/15pitch
Lines/Inch	3,4,6,8	3,4,6,8	3,4,6,8
Vertical Format Unit	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12
Expanded Characters	No	No	No
Loadable Fonts	Yes (Cartridge)	Yes	Yes
Ideographic Printing/ Graphics	No	Graphics	Graphics

Table 11-6. Characteristics of VS Remote Printers

Wang VS Model Number	2221V Matrix Printer	2231V Matrix Printer	2273V-1 BAND Printer	2233R Matrix Printer
Printing Speed	200 cps	120 cps	300 lpm	100 cps 120 cps
Size of Character Set	96 (Upper- and Lowercase)	96 (Upper- and Lowercase)	64 (Uppercase)	114
Chars/Inch (Horizontal Pitch)	10 (normal) 5 (expanded characters)	12 (normal) 6 (expanded characters)	12	10
Maximum Characters per Line	132	132	132	158
Lines/Inch	6	6	6 Standard 8 Optional	6,8
Vertical Format Unit	Paper Tape Channels 1 - 5	Paper Tape Channels 1 - 5	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12
Expanded Characters, Double Width	Yes	Yes	No	Yes
Loadable Fonts	No	No	No	No

(continued)

Table 11-6. Characteristics of VS Remote Printers (continued)

Wang VS Model Number	2235R Matrix Printer	2281WR Daisy Wheel
Printing Speed (cps)	180 220	30
Size of Character Set	114	96
Chars/Inch (Horizontal Pitch)	10	10,12,15
Maximum Characters per Line	132	132/10pitch 158/12pitch 198/15pitch
Lines/Inch	6,8	3,4,6,8
Vertical Format Unit	DAVFU Channels 1 - 12	DAVFU Channels 1 - 12
Expanded Characters	No	No
Loadable Fonts	No	No
Ideographic Printing/ Graphics	No	No

11.2 DATA BLOCKS

Data is passed from main memory to the printer in variable length blocks. The maximum data block length is 2048 bytes. Transfer of data is initiated by the SIO machine instruction. The associated IOCW indicates the type of block being passed to the printer, its address in main memory, and its length.

Listed below are the several types of blocks and their contents:

- Print Data Blocks -- Contain records to be printed, including text, graphics, and ideographic characters.

- Print Control Data Blocks -- Contain records used for controlling the printing of print data blocks. These records specify vertical and horizontal pitch, form length, standard font, and printer speed. The specified parameters remain in effect until reset by another Print Control Data Block.

- Font Data Blocks -- Contain font files. A default font may be loaded in the printer via font data blocks when the device is IPLed; other fonts may be loaded later when an application selects a font not already resident in the printer. Operating systems prior to Release 7.10 do not support font loading via font data blocks. Tables 11-1 to 11-6 indicate which printers support font loading.

- IPL Code Overlay Blocks -- Contain overlays of printer microcode. One overlay may be loaded in the printer via IPL Code overlay blocks when the device is IPLed; other overlays may be loaded later to support functions requested by an application. Operating systems prior to Release 7.10 do not support overlay loading via IPL code overlay blocks.

11.3 COMPRESSED RECORDS

Records in print data blocks and print control data blocks are transmitted to the printer in compressed form. A compressed record includes a two-byte record length (RL) field followed by a string of data whose format is dictated by the COMPRESS STRING machine instruction. The RL field indicates the compressed length of the data string plus the RL bytes. Figure 11-1 shows the format of a data block record.

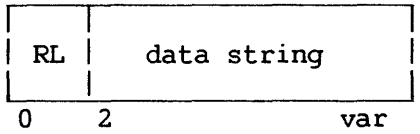


Figure 11-1. Format of Data Block Record

The data string of a record is made up of one or more substrings. Each substring includes a compression length (CL) byte followed by one or more data bytes, as shown in Figure 11-2.

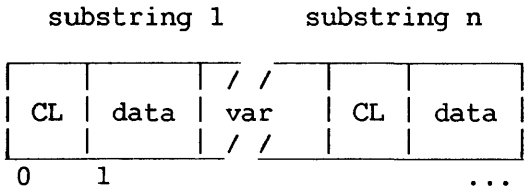


Figure 11-2. Format of Data String

When compression takes place, via the COMPRESS STRING instruction, a data byte repeated three or more times is compressed into a single byte. The high order bit of the CL byte indicates whether the following data has been compressed (1 = compression; 0 = no compression). The seven low order CL bits indicate the uncompressed length of the data that follows CL. When the high order bit of CL = 1, one data byte follows; and the seven low order bits indicate the number of times that byte is to be replicated by the printer. When the high order bit of CL = 0, up to 128 bytes of data follow; the seven low order CL bits hold a count of those bytes. For a full explanation of data compression and expansion, refer to descriptions of the COMPRESS STRING and EXPAND STRING instructions in Chapter 8 of this manual; also, to the description of compressed records in the Data Management System Reference.

Figure 11-3 shows the relationship of record, data string, and substring.

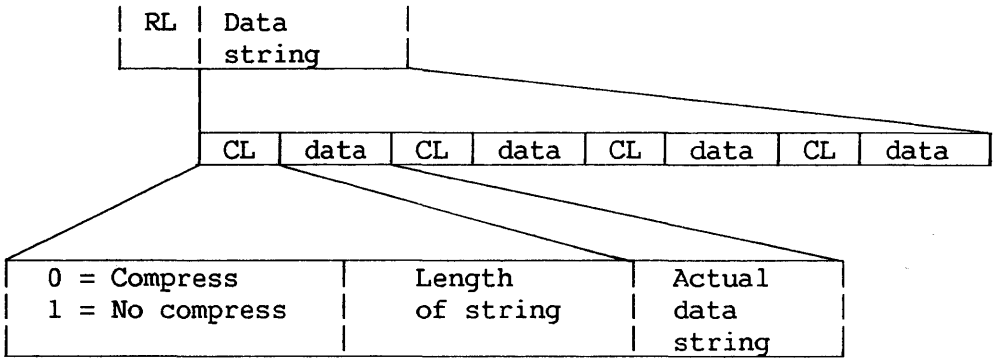


Figure 11-3. Elements of Compressed Record

11.4 POWER-UP IOCW

Not all systems support loading of fonts and IPL code overlays. On systems that do support these functions, a power-up IOCW is sent to each font/overlay loading printer after it has been powered up and loaded with microcode. Until a printer receives this IOCW, it does not issue IOSWs that request fonts or IPL code overlays. In response to a power-up IOCW, the font/overlay loading printer issues an IOSW that indicates whether the default overlay or font are to be sent to the printer during the IPL sequence.

11.5 FONT LOADING PROTOCOL

This section describes the exchange of IOCWs and IOSWs that takes place when fonts are loaded during printing.

When some record in a Print Data Block specifies a font to be used for printing, the following occurs. The printer determines whether the selected font has already been loaded. If necessary, it requests the font through an IOSW showing the following: normal completion; font request code (X'C3') in Byte 6; and a function code in Byte 7 that indicates whether the font to be loaded is the standard font or a new font. (The term "standard font" is defined in Section 11.13.8.) In addition, the IOSW indicates the number of print data records in the block processed before the font was requested and the number of print data bytes not yet processed.

When the printer requests the standard font, the system can determine the appropriate font number without interrogating the printer. The system then responds with one or more Font Data Block IOCWs, which define the memory location and size of blocks holding the requested font data.

When, instead, the printer requests a new font, the system responds with a Read Information IOCW to obtain the identifying number of the requested font. The data address field of this IOCW addresses a main memory location which is to receive from the printer the number of the requested font. After receipt of this information and an IOSW indicating normal completion, the system sends the requested font via Font Data Block IOCWs. The last Font Data Block IOCW is marked as such by a flag bit.

Each Font Data block successfully transmitted is acknowledged by an IOSW showing normal completion. After the last font data block is acknowledged, the system notifies the printer task, and stores at Location X'00' a normal completion IOSW. This IOSW shows the number of print records in the block processed before the font was requested and the number of print data bytes not yet processed. Using this information, the printer task issues a Write IOCW that causes printing of the print data block to be resumed.

When the system cannot honor a font loading request made through an IOSW, it responds with an Error IOCW that shows error code X'02', signifying "requested font unavailable". The printer then issues a Font Request IOSW that specifies the standard font. If another Error IOCW with error code X'02' indicates that the standard font is unavailable, the printer issues a Font Request IOSW for the default font. If the system cannot honor this request for the default font, it respnds with an Error IOCW that shows error code X'00'. A printer having resident fonts can continue printing without the default font; it issues an IOSW indicating normal completion. A printer without resident fonts issues an IOSW indicating error completion and showing an extended error code.

The format of the IOCWs and IOSWs mentioned in this summary are fully defined, respectively, in Sections 11.7 and 11.9.

11.6 IPL CODE OVERLAY LOADING PROTOCOL

This section describes the exchange of IOCWs and IOSWs that takes place when an overlay of microcode is loaded during printing.

Unlike fonts, IPL code overlays are not explicitly selected by an application; instead, an option requested by an application, such as printing of graphics, implicitly selects an IPL code overlay that supports the function.

Once an overlay has been implicitly selected, the printer requests the overlay through an IOSW showing the following: normal completion, overlay request code (X'C4') in Byte 6 and the identifying number of the overlay in Byte 7, the number of print data records in the block processed before the overlay was requested, and the number of print data bytes not yet processed.

After receipt of this information, the system sends the requested overlay via IPL Code Overlay Block IOCWs. The last IPL Code Overlay Block IOCW is marked as such by a bit flag. Each Overlay Data Block successfully transmitted is acknowledged by an IOSW showing normal completion. After the last font data block is acknowledged, the system notifies the printer task, and stores at location X'00' a normal completion IOSW. This IOSW shows the number of print records in the block processed before the overlay was requested and the number of print data bytes not yet processed. Using this information, the printer task issues a Write IOCW that causes printing of the print data block to be resumed.

When the system cannot honor an overlay loading request made through an IOSW, it responds with an Error IOCW showing an error code. If the printer can continue printing without the requested overlay, it issues an IOSW indicating normal completion; otherwise, the printer issues an IOSW indicating error completion and showing an extended error code.

The format of the IOCWs and IOSWs mentioned in this summary are completely defined, respectively, in Sections 11.7 and 11.9.

11.7 IOCW FORMAT

Figure 11-4 shows the format of the I/O Command Word (IOCW) passed to Wang serial printers.

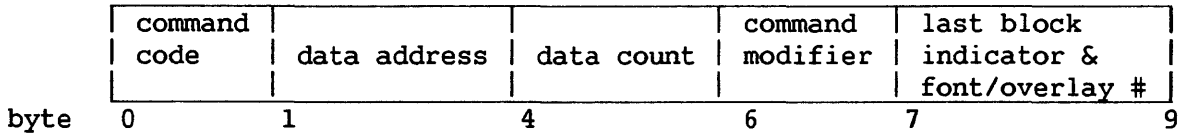


Figure 11-4. Printer IOCW

11.7.1 Command Code

Byte 0 of the IOCW, in conjunction with Byte 6, defines a request to the printer. Command code bits are defined as follows:

Bit	Meaning
0-1	00 = Reserved 01 = Read 10 = Write 11 = Control
2-4	Reserved; must be zero
5	0 = Data Address points to print data block 1 = Data address points to control data block
6	0 = Data address is a direct address 1 = Data address is an indirect address
7	0 = Translate characters into upper case 1 = Do not translate

The Read command is applicable only to printers that support font loading. Chain train printers accept only the Write command. Inapplicable commands cause the return of an invalid command IOSW.

11.7.2 Data Address

Bytes 1-3 address directly or indirectly (as indicated by Bit 6 of the command code) the main memory location of a data block to be processed by the printer.

11.7.3 Data Count

Bytes 4-5 indicate the number of bytes plus one, after any record compression, in the data block addressed by Bytes 1-3. This data count includes the block length indicator that comprises the first two bytes of every data block. The value of the data count and the block length indicator must agree, or the command is suppressed with error and incorrect length indications.

In IOCWs addressing blocks of compressed data (i.e., print data blocks or print control data blocks), allowable data counts range from 6 to 2048 bytes. The one exception to this rule is presented by a control data block that resets the system defaults for all options controlled by the block. The length of a control data block so used is 4 bytes, which, in this one instance, is an allowable value for the IOCW data count. In IOCWs pertaining to font or overlay data blocks, allowable data counts range from 1 to 2048 bytes.

11.7.4 Command Modifier

Command modifier codes in Byte 6 of the IOCW modify commands specified by Byte 0. In the case of Read, Write, or Control commands involving data blocks, the command modifier codes indicate the type of data block involved. The following command modifier codes are defined:

X'00' = Print data block or control data block
X'C3' = Font data block
X'C4' = IPL code overlay data block
X'C5' = Power-up IOCW (no data block)
X'C6' = Reserved
X'C7' = End of Job IOCW (no data block)
X'C8' = Reserved for ideographic functions
X'CA' = Reserved for ideographic functions

11.7.5 Last Block Indicator and Font/IPL Code Overlay Number

Bit 0 of Byte 7 is set to one when the block addressed by Bytes 1-3 is the last block in the file.

Bits 1-5 of Byte 7 are reserved. Bits 6-7 of Byte 7 and all bits of Byte 8 form a 10-bit identifying number of a font or overlay being transmitted to the printer.

11.8 IOCW TYPES

Valid values for the IOCW command code (Byte 0) and command modifier (Byte 6) fields define seven types of IOCWs, which are described in the following sections.

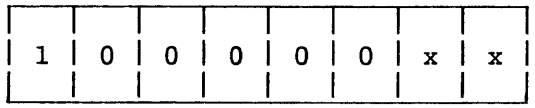
NOTE

All seven IOCWs are sent to the printer by means of the SIO machine instruction.

11.8.1 Print Data Block IOCW

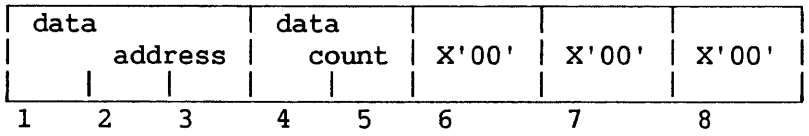
This IOCW requests printing of print data block records.

Byte 0



Bit 5 = 0 to indicate a print data (rather than control data) block. Bits 6 and 7 can be set to indicate indirect addressing and uppercase translation.

Bytes 1-7

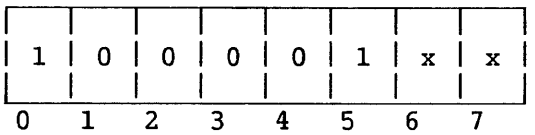


A value of X'00' in Byte 6 indicates that this IOCW initiates processing of the print data block described by Bytes 1-5.

11.8.2 Control Data Block IOCW

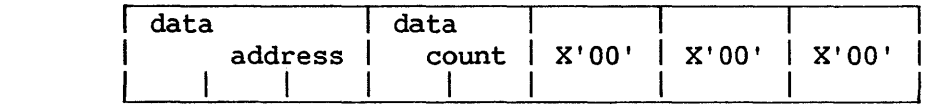
This IOCW transfers to a printer information used to control printing.

Byte 0



Bit 5 = 1 to indicate a control data (rather than print data) block. Bit 6 can be set to indicate indirect addressing; Bit 7 is ignored.

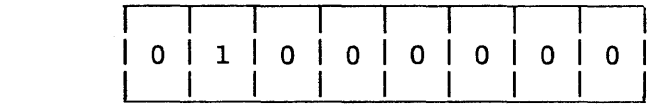
Bytes 1-7



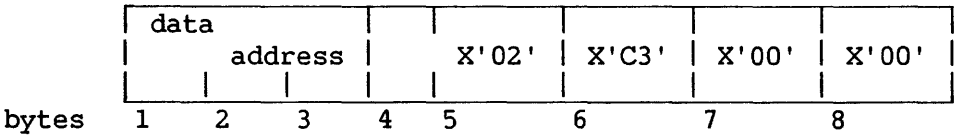
11.8.3 Read Information IOCW

This IOCW is sent to a device after it has requested the loading of a new font. It is a read request for 2 bytes of data that specify the identifying number of the font to be sent to the device.

Byte 0



Bytes 1-7

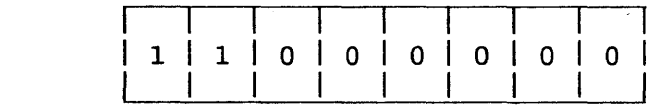


A value of X'C3' in Byte 6 when Byte 0 = X'40' indicates a request for a font identifier. Bytes 1-3 address a main memory location which is to receive a 2-byte font identifier from the printer. Byte 5, the second byte of the data length field, indicates the 2-byte length of the font identifier.

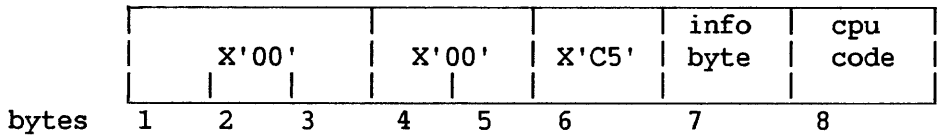
11.8.4 Power-Up IOCW

This IOCW indicates whether the system supports IPL overlay code and font loading.

Byte 0



Bytes 1-7



The value of X'C5' in Byte 6 identifies this IOCW, distinguishing it from other types of IOCWs whose command code (Byte 0) is also X'C0'.

A value of X'00' in Byte 7 indicates that font and IPL code overlay loading are not supported. A value of X'01' indicates that requests for font/IPL code overlay loading are supported.

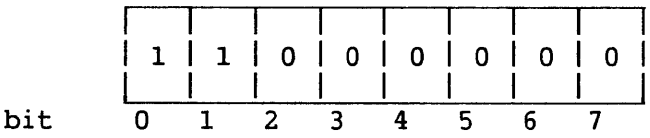
The CP codes reported in Byte 8 identify the model of the VS system as follows:

CP Code	VS Model
X'05'	VS15, VS45
X'07'	VS65
X'04'	VS85, VS90, VS100
X'08'	VS300

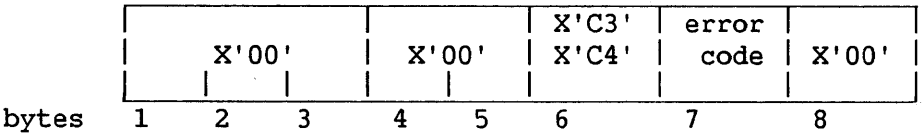
11.8.5 Error IOCW

This IOCW is issued to indicate that the system cannot honor a printer's request for a font or IPL code overlay.

Byte 0



Bytes 1-7



The value of X'C3' in Byte 6 when Byte 0 = X'C0' signifies that the system is unable to load a requested font. In this case, Byte 7 shows one of the following error codes:

X'00' = Default file not found
X'01' = Error reading file
X'02' = Requested font unavailable

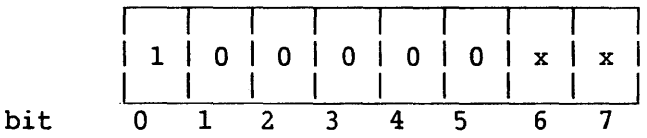
The value of X'C4' in Byte 6 when Byte 0 = X'C0' signifies that the system is unable to load a requested IPL code overlay. In this case, Byte 7 shows one of the following error codes:

X'00' = Code overlay not found
X'01' = Error reading overlay file

11.8.6 IPL Code Overlay Block IOCW

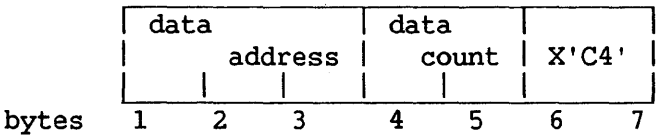
This IOCW allows overlays of microcode to be transferred to the printer in IPL Code Overlay Blocks.

Byte 0



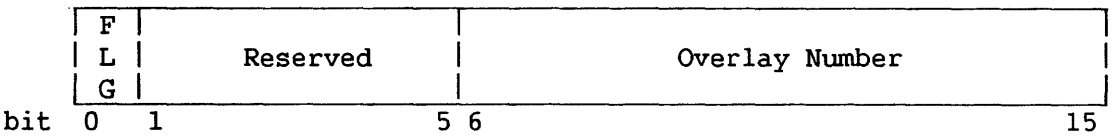
Bit 6 can be set to indicate indirect addressing; Bit 7 is ignored.

Bytes 1-6



A value of X'C4' in Byte 6 indicates that an IPL code overlay block is associated with this IOCW.

Bytes 7-8

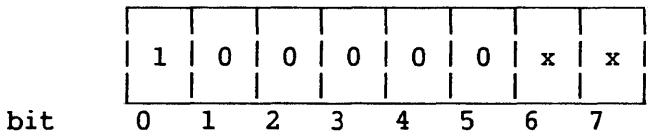


Bit 0 of Byte 7 is a flag bit that, when set to 1, indicates the last block of the file. Bits 1-5 of Byte 7 are reserved for future use. Bits 6-7 of Byte 7 and Bits 0-7 of Byte 8 hold the number of the overlay being loaded.

11.8.7 Font Data Block IOCW

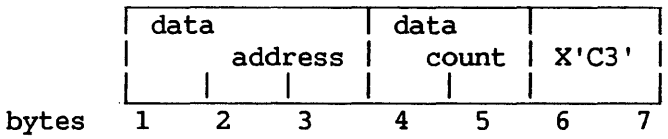
This IOCW allows font files to be transferred to the printer in font data blocks.

Byte 0



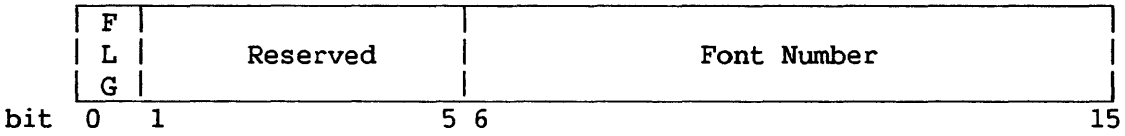
Bit 6 can be set to indicate addressing; Bit 7 is ignored.

Bytes 1-6



A value of X'C3' in Byte 6 indicates that a font data block is associated with this IOCW.

Bytes 7-8

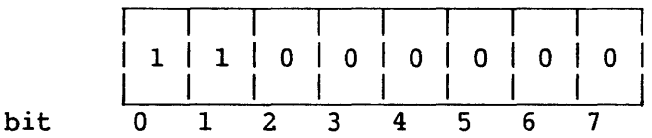


Bit 0 of Byte 7 is a flag bit that, when set to 1, indicates the last block of the file. Bits 1-5 of Byte 7 are reserved for future use. Bits 6-7 of Byte 7 and Bits 0-7 of Byte 8 hold the number of the font being loaded. This is not necessarily the number of the requested font; when unable to load the requested font, the system may load a default font instead.

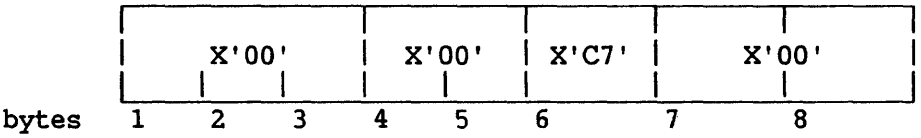
11.8.8 End of Job IOCW

This IOCW is issued to the printer after the printer has acknowledged, with a normal completion IOSW, the last print data block of a print file.

Byte 0



Bytes 1-7



A value of X'C7' in Byte 6 identifies this IOCW, distinguishing it from other IOCWs with a command code of X'C0'.

11.9 PRINTER IOSW

Figure 11-5 shows the format of the I/O status word (IOSW) passed from the printer to the system.

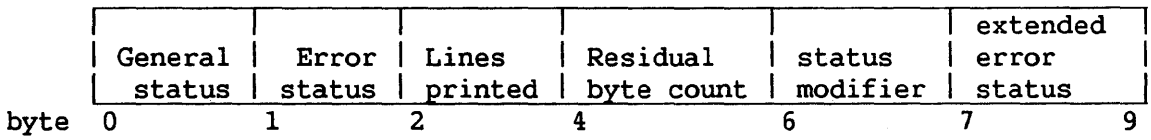


Figure 11-5. Printer IOSW Format

11.9.1 General Status Byte

Bits in the the General Status byte (Byte 0) have the following significance when set to 1:

Bit	Mnemomic	Meaning
0	IRQ	Intervention required
1	NC	Normal completion
2	EC	Error completion
3	U	Unsolicited
4	PC	IOP now ready (VS100 only)
5	DAR	Data area early release
6-7		Reserved; must be zero

The printer sets the IRQ, NC, EC, U, and DAR bits; the I/O processor sets the other bits of the general status byte.

These error status codes are standard in VS I/O protocol and are more fully explained in Chapter 9.

11.9.2 Error Status Byte

Bits in the Error Status byte (Byte 1) have the following significance when set to 1:

Bit	Mnemomic	Meaning
0	IC	Invalid command
1	MPE	Memory parity error
2	MAE	Memory address error
3	DM	Device malfunction
4	DAM	Memory or device damage
5	IL	Incorrect length
6	DP	Device microcode required
7	PP	Peripheral processor microcode required

The IC, DM, and IL bits are set by the printer; the other bits of the error status byte are set by the I/O processor.

These error status codes are standard in VS I/O protocol and are more fully explained in Chapter 9.

11.9.3 Lines Printed Bytes

IOSW Bytes 2-3 indicate the number of records in the current print data block that have been printed, except when the General Status byte indicates an unsolicited interrupt, in which case Bytes 2-3 are zero.

11.9.4 Residual Count Bytes

Bytes 4-5 report the number of bytes in the data block remaining to be processed after processing has been interrupted by one of the following:

- An error condition
- A Halt I/O instruction
- A request by the printer for a font or IPL code overlay
- A request by the printer for suspension of data transmission

11.9.5 Status Modifier Byte

This byte can indicate a busy state or a request for a font or IPL code overlay. Values for the byte are defined as follows:

Code	Meaning
X'00'	Normal or error completion of an IOCW order
X'C3'	Request font
X'C4'	Request IPL code overlay
X'C8'	Reserved for ideographic function
X'C9'	Printer busy; no further data accepted until requested
X'CA'	Reserved for ideographic function

11.9.6 Extended Error Status Byte

The contents of Byte 7 depend on whether the IOSW is issued to request a font, to request an overlay, to report an error, or to answer a Power-Up IOCW.

Font Request

When Byte 6 contains X'C3' (request for font loading), Byte 7 contains one of the following function codes:

Code	Meaning
X'00'	Load default font
X'02'	Load new font

Overlay Request

When Byte 6 contains X'C4' (request for IPL overlay code loading), Byte 7 contains the number of the requested overlay.

Error Report

Byte 7 can contain an extended error status code that signals the system to abort the current I/O order or IPL, depending on the particular code. The IOSW showing this code properly follows an Error IOCW (which reports that a requested font/IPL code overlay cannot be loaded) or a Power-Up IOCW (which indicates that font/IPL code overlay loading is not supported). In this IOSW, the general status must be error completion, the error status must be zero, and Byte 7 set to one of these extended error status codes:

Code	Meaning
X'01'	Abort order because overlay is required
X'02'	Abort order because font is required; no substitute possible
X'03'	Abort IPL because printer requires support for loading of fonts or IPL overlay code
X'04'-X'05'	Reserved for ideographic functions

NOTE

All codes that set the high order bit of Byte 7 ON are reserved for ideographic functions.

Power-Up IOSW

In response to a Power-Up IOCW that indicates support of font/IPL code overlay loading, the printer issues an IOSW indicating whether Overlay 0 or the default font is to be loaded. Byte 6 is zero; Bits 0-1 of Byte 7 are defined as follows:

Bit	Meaning when Set to 1
0	Load Overlay 0
1	Load default font

11.9.7 Suspend and Resume IOSW

When the printer, because of a busy condition, wishes to suspend transmission of data, it issues an IOSW with a code of X'C9' in Byte 6. The printer can issue a Suspend IOSW after receipt of an End-of-Job IOCW or data block. Thus, the IOSW can suspend data transmission that is in progress or anticipated. When the IOSW is issued to suspend data transmission in progress, the residual data count shows the number of bytes in the data block remaining to be processed.

To cancel the Suspend IOSW, the printer sends an unsolicited IOSW with the U bit set in Byte 0, a Ready code of X'60' in Byte 2, and one of the following function codes in Byte 7:

Code	Meaning
X'01'	Resume I/O
X'03'	Error status in Byte 1
X'05'	Intervention required

When the system receives a Suspend IOSW and then the unsolicited IOSW just described, it can resume data transmission at the point indicated by the residual count field of the Suspend IOSW.

Between the reception by the system of a Suspend and Resume IOSW, the printer is not subject to timeout.

11.10 PRINT DATA BLOCK

The print data block is used to transmit records of printable data. This data can be in the form of character strings, graphics data, or ideographic data. A single block may contain more than one of these data types.

A block consists of 2 block length bytes followed by an integer number of records. The block length count equals record bytes, plus the two block length bytes, plus 1. The range of valid block length counts, in decimal, is 6 to 2048 inclusive. The block length count and the data count of the associated IOCW must have the same value.

11.10.1 Print Data Records

In general, one record represents one line of printed data. In the first substring of a record, the compression length (CL) byte is followed by two to six print control bytes. This is the only occurrence of print control bytes in a record.

Figure 11-6 illustrates the format of a print data block, showing the relationship of block, records, substrings, and print control bytes. In the illustrated substring, the printable data following the print control bytes happens to be uncompressed. If the printable data were compressed, it would be preceded by a CL byte indicating compression (i.e., it would belong to a second substring.).

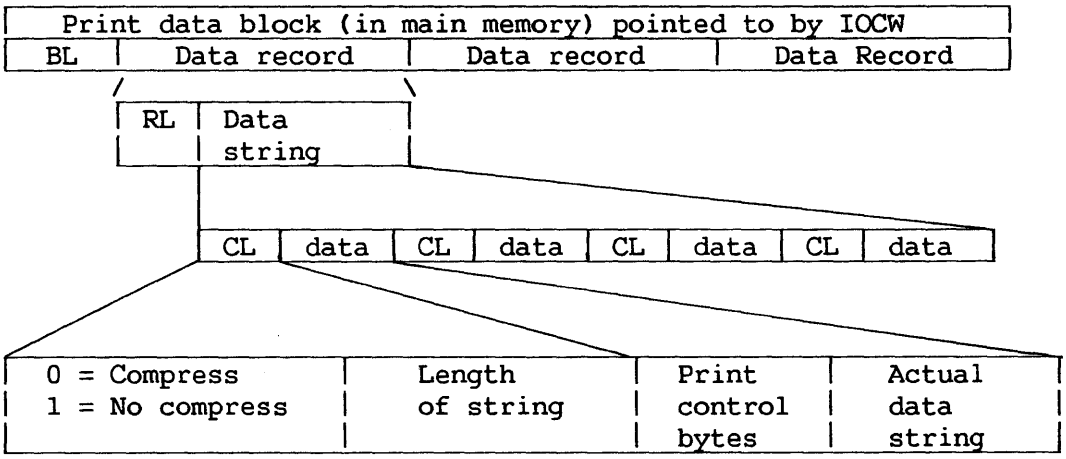


Figure 11-6. Format of Print Data Block and Record

11.11 PRINT CONTROL BYTES

Each print data record includes from two to six print control bytes (PCBs). These control print functions at the record level, whereas print functions at the block level are controlled by the Print Control Data Block.

Bit settings of PCBs control the following functions:

- Vertical Spacing
- Hardware alarm
- Double width printing
- Line feed direction
- Font selection
- Graphic printing selection
- Ideographic printing selection
- Sheet feeder/bin selection
- Ribbon selection

11.11.1 Chain Bits

Two PCBs are required to control vertical spacing. The presence of additional PCBs is indicated to the printer by setting a chain bit (Bit 7) as follows: A third PCB (PCB 3) is indicated by the chain bit of the first PCB; PCBs 4 through 6 are each indicated by the chain bit of the preceding PCB; there is no chain bit in PCB 2.

Although there is currently a maximum of 6 PCBs per record, the VS architecture allows for the chaining of an indefinite number. In the future, additional PCBs may be defined to support new functions. PCBs 7 and 8 are reserved for ideographic functions.

11.11.2 Unsupported Functions

Generally, when PCBs specify a function not supported by the receiving printer, the printer ignores the selection as if it had not been made; in a few cases, the printer generates an Invalid Command IOSW. The following descriptions of PCBs assume that unsupported functions are ignored, and mention only those instances that generate an error IOSW.

11.11.3 PCB Bit Definitions

PCB bits are defined as shown below. Explanations of the functions selected by PCB bits are provided in later sections.

PCB 1

Bit	Meaning
0	0 = Space the number of lines specified by PCB 2 1 = Use skip channel specified by PCB 2 for line spacing
1	0 = Line space before printing 1 = Line space after printing
2	0 = Normal width characters 1 = Double width characters
3	0 = No hardware alarm 1 = Activate hardware alarm
4	0 = Select bin/feeder 2 1 = Select bin/feeder 1
5	0 = Use normal ribbon 1 = Use alternate ribbon
6	Reserved for future use
7	0 = There are only two PCBs 1 = A third PCB follows the first two

PCB 2

Bit	Meaning
0	0 = Line feed in positive direction 1 = Line feed in negative direction
1-7	This bit is significant only when Bit 0 of PCB 1 = 0 Lines to be skipped (if Bit 0 of PCB 1 = 0) or skip channel number (if Bit 0 of PCB 1 = 1)

PCB 3

Bit	Meaning
0-2	000 = Use standard font (or left print wheel) 001 = Use font 1 (or right print wheel) 010 = Use font 2 011 = Use font specified in PCBs 4-5 100 = Interpret this record as graphics data 101 thru Reserved for ideographic functions 111
3	Reserved for future use
4-6	bin number (if Bit 4 of PCB 1 = 0)
7	0 = This is the last PCB 1 = There is a fourth PCB

PCB 4

Bit	Meaning
0-6	Most significant seven bits of requested font number
7	Must be 1 (i.e., this PCB must be followed by PCB 5)

PCB 5

Bit	Meaning
0-6	Least significant seven bits of requested font number
7	0 = This is the last PCB 1 = There is a sixth PCB

PCB 6

Bit	Meaning
0-6	Reserved
7	Must be zero

11.11.4 PCB 1 Options -- Double Width Characters

The double-width character function affects printing of both ideographic and font characters. If this option is selected, all printable data contained in the current record is printed in double width. If both double- and normal-width characters are desired on the same line of print, it is necessary to create two print records and then to overstrike one with the other.

11.11.5 PCB 1 Options -- Sheet Feeder/Bin Select

PCB 1 Bit 4 applies to printers that receive sheets from twin sheet feeders, twin bins, or several bins. PCB 1 Bit 4 can be used to select bin 1 or 2. Setting the bit to 1 selects bin 1; setting the bit to 0 selects bin 2 or any bin specified by PCB 3 Bits 4-6. Thus, when the first two PCBs are present, either of two bins may be selected; when the first three PCBs are present, one of seven bins may be selected. (Currently, no printer supports more than three bins.) The interaction of bin selection bits when three PCBs are present is shown below.

PCB 1 Bit 4	PCB 3 Bits 4-6	Bin Selected
Bit Values:		
1	ignored	Bin 1
0	0	Bin 2
0	1-6	Specified bin
0	7	Highest numbered bin

Selecting a sheet feeder or bin has no effect unless PCB 1 also specifies "use skip channel" and PCB 2 specifies skip channel 1. That is, sheet feeder/bin selection is ignored unless PCBs 1 and 2 explicitly specify top of form. In the absence of an explicit top of form, the printer uses the sheet feeder/bin last selected. In the absence of an explicit top of form and a previous selection, the printer uses the default, which is sheet feeder/bin 2.

Out of Paper Action

If the selected or default sheet feeder/bin is empty, the printer issues an "intervention required" operator message, deselects, and waits to be reselected. There is one exception: When a bin that has been selected as the highest numbered bin (PCB 1 selection bit = 0 and PCB 3 selection bits = 7) is empty, the printer attempts to use the bin with the next lower number.

Sheet Feeder/Bin Designation

Twin sheet feeders/bins are numbered as follows: The sheet feeder nearer the cable side of the printer is sheet feeder 2. The lower twin bin is bin 2.

The bins on laser printers are numbered as follows:

LPS-8	Bin 3 = the standard bin 2 = the higher optional bin 1 = the lower optional bin
LIS-12	Bin 2 = the higher bin 1 = the lower bin
LIS-24	Bin 3 = the 1500 sheet bin 2 = the higher of the two remaining bins 1 = the lower of the two remaining bins

11.11.6 PCB 2 Options

Negative Line Feed

Specifying negative, or backward, line feed has no effect when a skip channel is being used (i.e., when Bit 0 of PCB 1 = 1). When the number of lines to be skipped (specified by Bits 1-7 of PCB 2) is zero, the linefeed bit (Bit 0 of PCB 2) is ignored.

11.11.7 PCB3 Options -- Font Specification

The setting of Bits 0-2 determines the font used to print the current record. Some bit settings apply to a particular type of printer; bit settings that apply to more than one type of printer are interpreted according to the type involved.

For purposes of explaining font specification via PCB 3, printers may be grouped into the following three types:

1. Printers supporting two fonts in the form of two print wheels, referred to logically as fonts 0 and 1.

2. Printers supporting two fonts held in their microcode, referred to logically as fonts 1 and 2.

3. Printers supporting more than two fonts held in their microcode or in cartridges.

The remaining paragraphs of this section describe settings of PCB 3, Bits 0-2 that are defined for font selection.

Bit Setting 000

A bit setting of 000 applies to Types 1, 2, and 3. When applied to type 1, this setting signifies "use the left print wheel" (wheel 1, logical font 0). When applied to Types 2 and 3, this setting signifies "use the standard font."

The standard font is designated by means of a control data block, as explained in Section 11.13.8. If not designated by a control data block, the standard font is that specified as the default in the font catalog. In the absence of PCB 3 from a record, the font used for printing the current record is the designated or defaulted standard font.

Bit Setting 001

A bit setting of 001 applies to printer Types 1 and 2. When applied to Type 1, this setting means "use right print wheel" (wheel 2, logical font 1). When applied to Type 2, this setting means "use logical font 1."

Bit Setting 010

A bit setting of 010 applies only to printers of Type 2 and means "use logical font 2."

Bit Setting 011

A bit setting of 011 applies to printers of Type 2 and 3, and signifies "use the font specified in PCBs 4-5". This setting is the only means of selecting a font for printers of Type 3. It is the preferred method of selecting a font for printers of Type 2 that are configured in a fontloading system because on a fontloading system, the actual font numbers associated with logical fonts 1 and 2 can change during the printing of a print file.

11.11.8 PCB 3 Options -- Graphics Printing

Graphics printing is described in Section 11.12.

11.12 GRAPHICS PRINTING

A graphics record in a print data block consists of three print control bytes, followed by at least one graphics command. Raster data is an argument of the DRAW RASTER and DRAW PIXEL commands.

Graphics printing is specified by setting Bits 0-2 of PCB 3 to binary 100. When graphics printing is specified, the only other significant PCB bits are those controlling hardware alarm and skip channels. Other bits are ignored.

11.12.1 Graphics Protocol

The printer buffers rasterized data in units of the raster line, which is defined as the equivalent in bits of the pixels on one scan line of the CRT. It is an axiom of graphics protocol that a command pertains only to one raster line. Accordingly, rasterized data supplied with a command belongs to the current raster line only; it is not carried from the current raster line to the next. The printer discards rasterized data in a DRAW command that exceeds the length of a raster line. Raster data of multiple DRAW commands in immediate succession is concatenated until the maximum raster line length is reached. Any remaining data in the last concatenated command is discarded. Similarly, a MOVE command or series of concatenated MOVEs cannot move the virtual cursor beyond the end of a single raster line. Commands involving vertical moves from one raster line to the next (MOVE DOWN, LINE FEED, CARRIAGE RETURN/LINE FEED) implicitly signal the end of the current raster line.

When the printer buffer is filled with raster lines, the buffer is printed. Printing can be forced before the buffer is full by any of the following:

- A Print Data record whose PCBs specify skip to channel 1 (top of form)
- A print data record whose PCB 3 specifies text or ideographic printing.
- A MOVE DOWN graphics command that specifies a move of enough raster lines so that the sum of currently buffered raster lines plus the vertical move is greater than the number of raster lines buffered before printing.

11.12.2 Graphics Command Syntax

The first two bytes of each command hold a count of bytes in the command, including the 2-byte count. A count of 0 or 1 is invalid; the result is undefined. A count of 2 is valid; it implies that the next two bytes hold another length count. The second and third bytes of a command consist of an operation code. The remaining bytes of the command, if any, vary in format between commands. The maximum length of a command is the maximum record length supported by the system.

11.12.3 Graphics Commands

In the following command descriptions, parenthetical numbers indicate field length in bytes. Values are decimal.

NOTE

Some printers capable of graphics printing support only a subset of the commands described in this section. To ascertain the commands supported by a printer, refer to the user manual describing that printer.

Define Color

Fields

Field	Value	Description
length (2)	7	Byte count of command
opcode (1)	10	Define color
index (1)	0 - 255	Index to color in map
red intensity (1)	0 - 255	255 = full intensity
green "	"	"
blue "	"	"

Effect: Defines a color, addressed by the index parameter, in a red/green/blue (RGB) color model map.

Default: There are two system-defined default indexes:

index 0 = the normal background for the device (usually white)

index 255 = the complement of normal device background

Notes: A maximum of 255 colors can be simultaneously defined. Both index 0 and index 255 may be redefined by the DEFINE COLOR command.

The RGB color model is additive, but mixing pigments on a printer ribbon does subtractive coloration. A pixel written in both red and green is meant to be yellow. A printer should only strike both red and green pigments at the pixel if that yields the best available approximation to yellow.

Select Color

Fields:

Name	Value	Description
length (2)	4	Byte count for command
opcode (1)	11	Select color
index (1)	0 - 255	Index to an RGB color in map

Effect: Selects the color to be applied to "1" bits (i.e., bits set to 1) in raster format image data in DRAW RASTER commands.

Default: Index = 255

Notes: If the index is not defined, i.e. is neither a system default definition nor previously defined by the DEFINE COLOR command, index 255 is selected. Thus, unless index 255 has been redefined from its system default, the complement to normal background color is selected. Since printer paper is usually white, black would be struck for each "1" bit in image data passed in a DRAW RASTER command.

Draw Raster

Fields:

Name	Value	Description
length (2)	5 - 32767	Byte length of command
opcode (1)	20	Draw raster formatted data
pad count (1)	0 - 7	# pad bits in last byte
data byte 1	0 - 255	1's = draw, 0's = skip
byte 2	"	"
...	"	"
byte n	"	"

Effect: The data is buffered or drawn in the currently selected color starting at the location of the virtual cursor which ends one pixel beyond the last data (not pad) pixel.

Default: None

Notes: This command assumes that the data represents raster format pixels, whereby 1s are drawn, 0s are skipped.

The maximum value for a given printer is limited by the size of its buffer.

Draw Pixel

Fields

Name	Value	Description
length (2)	4 - 32767	Byte count for command
opcode (1)	21	Draw pixel
data byte 1	0 - 255	Index for pixel 1
byte 2	"	pixel 2
...	"	...
byte n	"	pixel n

Effect: Draws each pixel in the color given by the index specified for that pixel starting at the current position of the virtual cursor. The cursor ends one pixel beyond the last one drawn.

Default: None

Notes: If an index has not been previously defined, the color at index 255 is used (initialized to the complement of background).

Draw Vectors

Fields:

Name	Value	Description
length (2)	4 - 32767	Byte count for command
opcode (1)	22	Draw vectors
data byte 1	0 - 255	Index for pixel 1
byte 2	"	pixel 2
...	"	...
byte n	"	pixel n

Effect: Processes the data as a sequence of positioning commands (such as MOVE or DRAW), and non-positioning commands (such as TEXT, DEFINE PEN, etc.).

Default: None

Notes: The specific implementation of this command is a function of the printer or interface to which it is connected.

Move Right

Fields:

Name	Value	Description
length (2)	5	Byte count for command
opcode (1)	30	Move right
pixel count (2)	0 - 32767	Number of pixels to move

Effect: Moves the virtual cursor to the right by the counted number of pixels.

Default: None

Notes: If the cursor is moved beyond the printer's buffer extent, the result is truncation to the last pixel location in the buffer.

Move Left

Fields:

Name	Value	Description
length (2)	5	Byte count for command
opcode (1)	31	Move left
pixel count (2)	0 - 32767	Count of pixels to move

Effect: Moves the virtual cursor to the left by the counted number of pixels.

Default: None

Notes: Truncates at starting location of pixels.

Move Down

Fields:

Name	Value	Description
length (2)	5	Byte count for command
opcode (1)	32	Move down
line count (2)	1 - 32767	Number of lines to move down

Effect: Does an implicit end of raster and moves the virtual cursor down the requested number of raster lines. This may cause the raster buffer to be printed and the paper to be moved.

Default: None

Notes: A count of "1" renders this command identical in function to the LINE FEED command.

Carriage Return

Fields:

Name	Value	Description
length (2)	3	Byte count for command
opcode (1)	33	Carriage return

Effect: Returns the virtual cursor to the left-hand side start location for pixels.

Default: None

Notes: None

Line Feed

Fields:

Name	Value	Description
length (2)	3	Byte count for command
opcode (1)	34	Line feed

Effect: Does an implicit end of raster and moves the virtual cursor to the next raster line.

Default: None

Notes: None

Carriage Return Line Feed

Fields:

Name	Value	Description
length (2)	3	Byte count for command
opcode (1)	35	Carriage return line feed

Effect: Returns the virtual cursor to the right-hand side starting pixel location, does an implicit end of raster, and skips to next raster line.

Default: None

Notes: This command does an implicit end of raster command as does any other vertical move.

11.13 CONTROL DATA BLOCK

The control data block contains data that controls the printing of whole print data blocks. (Print Control Bytes, by contrast, control the printing of a record within a print data block.) The block is passed from main memory to the printer by means of an IOCW that describes the type, location, and length of the block.

Control data block records are compressed. One print control block can include records relating to the control of different functions, and these records may be of different length.

Control data blocks control the following functions:

- Vertical pitch
- Horizontal pitch
- Form control (via Direct Access Vertical Format Unit [DAVFU])
- Standard font
- Printer speed

The format of the print control data block is shown in Figure 11-7.

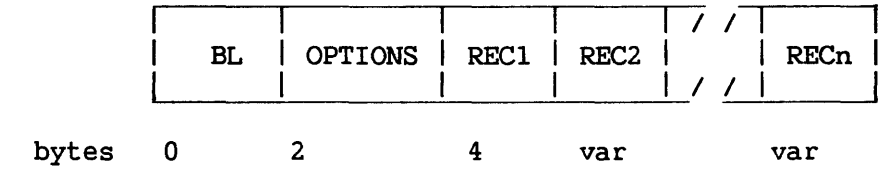


Figure 11-7. Format of the Control Data Block

11.13.1 Block Length Bytes

The two block length (BL) bytes hold a count that equals the record bytes plus the two block length bytes plus one. The range of valid block length counts is 4-2048. The value of the count must equal the value of the data count in an associated IOCW.

11.13.2 Options Bytes

The options bytes serve as a function mask whose bits, when set to 1, select functions whose parameters are specified in a record following the options bytes. Bits in the options byte are defined as follows:

Bit	Function
0	Reset parameters for all functions to default value.
1	Vertical pitch
2	Horizontal pitch
3	Direct Access Vertical Format Unit (DAVFU)
4	Standard Font
5	Printer speed
6	Reserved for ideographic function
7-15	Reserved for future use

There must be a record following the options bytes for each option bit that is set, and the order of the records must be the same as the order of the set option bits.

When a printer is initialized, default parameters are set for all of the functions listed above. These defaults remain in effect until expressly altered. Some malfunctions (e.g., loss of power) can cause the default values to be reinstated. The user may reestablish default values for all parameters by setting Bit 0 of the options field to 1, and all other bits of the field to 0. If Bit 0 and any other bit are set, an error completion is returned from the printer.

11.13.3 Format of Control Data Block Records

Control data block records are compressed before transmission to the printer. As explained in Section 11.3, each compressed record includes a 2-byte record length field, followed by one or more data substrings. Each data substring includes a Compression Length (CL) byte followed by data.

11.13.4 Vertical Pitch Record

The Vertical pitch record, shown in Figure 11-8, specifies the number of printed lines per inch.

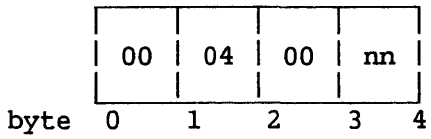


Figure 11-8. Format of Vertical Pitch Record

Bytes 0-1, the RL field, specify a record length of 4 byes. Byte 2, the CL field, specifies no compression. Byte 3, the data byte, specifies lines per inch (lpi) by one of the following codes:

00 = default; usually 6 lpi
01 = 8 lpi
02 = 3 lpi
04 = 4 lpi

The printer initializes the horizontal pitch to the default value when the printer is powered on or in the event of certain malfunctions.

After processing this record, some printers redefine top of form as the current line; other printers recompute the current line number, using the lpi value supplied by the record.

Sending this record to a printer whose vertical pitch is not programmable results in the return of an IOSW with an Invalid Command error.

If the printer supports vertical pitch selection, but not the particular pitch selected by this record, the printer adopts the default pitch and returns an IOSW without an error. The printer takes the same action upon receiving a code that is not one of those listed above.

11.13.5 Horizontal Pitch Record

The horizontal pitch record specifies the number of printed characters per inch (cpi). This record is the same in format as the vertical pitch record. The values of the record fields are also the same, except for the codes held by Byte 3, which are defined as follows:

00 = default (usually 10 cpi)
01 = 12 cpi
02 = 15 cpi
03 = 16.5 cpi

The printer initializes the horizontal pitch to the default value when the printer is powered on or in the event of certain malfunctions.

Sending this record to a printer whose horizontal pitch is fixed may result in the return of an IOSW with an Invalid Command error.

Printers whose horizontal pitch is a function of the font selected ignore this record and do not report an error in the IOSW.

If the printer supports horizontal pitch selection, but not the particular pitch selected by this record, the printer adopts the default pitch and returns an IOSW without an error. The printer takes the same action upon receiving a code that is not one of those listed above.

11.13.6 Printer Speed Record

The printer speed record specifies speed of printing and affects its quality; at slower print speeds, the quality of print increases.

This record is the same in format as the vertical pitch record. The values of the record fields are also the same, except for the codes held by Byte 3, which are defined as follows:

00 = printer's maximum speed (default)
01 = approximately 1/2 of printer's maximum speed

Codes other than those listed are interpreted by the printer as the default.

The printer initializes its speed to the default value when the printer is powered on or in the event of certain malfunctions.

Printers whose speed is a function of the font selected ignore this record and do not report an error in the IOSW.

11.13.7 Direct Access Vertical Format Unit Record

The Direct Access Vertical Format Unit (DAVFU) record simulates a 12-channel paper tape whose channels are defined by the DAVFU record.

DAVFU Record Format

DAVFU data follows the record length and compression bytes of the DAVFU record. This data has the same format as a table in the memory of a printer supporting DAVFU. In this table, two-byte rows represent the rows of a 12-channel tape; there are as many pairs of bytes as lines on a page. The setting of each bit in a two-byte row represents a hole or the absence of one in a channel (1 = hole; 0 = no hole). Since a pair of bytes equals 16 bits and only 12 are needed to represent channels, the setting of four bits (Bits 0, 1, 8, and 9) are ignored.

When the printer is initialized, and in the event of certain malfunctions, the DAVFU table receives the default values shown in Figure 11-9.

Channel Bit	b0	b1	12 b2	11 b3	10 b4	9 b5	8 b6	7 b7	b8	b9	6 10	5 11	4 12	3 13	2 14	1 15
Line 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
Line 1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line 2	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line 3	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line 4	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line 5	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line 6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Line 7	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line 8	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line 9	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line 10	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
Line n	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0

Figure 11-9. Default Values of DAVFU Table

Default DAVFU table values are changed by setting Bit 3 of the option field to 1 and providing a record whose data portion duplicates the format of the table shown in Figure 11-9. The data portion is received by the printer in compressed form; after the data is expanded, it replaces the existing values in the table.

In a DAVFU record, Channel 1 must retain the default definition as top of form. On band printers, the first position of Channel 2 must retain the default definition of 0 to allow proper operation of the Page Eject function. Otherwise, there are no constraints on the channel definitions supplied by DAVFU records.

Form Length

The length of the DAVFU record indicates the number of lines per form (form length). Form length, together with the vertical pitch (as indicated by the vertical pitch record), defines for the printer the dimensions of the paper used for printing. The manner in which DAVFU record length indicates form length depends on the manner in which the printer receives paper.

Printers receive paper as a continuous form or as sheets from twin sheet feeders, twin bins, or multiple bins.

Twin sheet feeders, supported by daisy wheel printers and the 5577 matrix printer, hold one paper size: 8.5 x 11 inches.

Twin bins, supported by the 5577 matrix printer and the DW/55 daisy wheel printer, hold paper of various dimensions, some fractional. Each bin in a pair holds paper of the same dimension; however, the paper in bin 2 may be fed broadside (for landscape printing) rather than lengthwise (for portrait printing).

A laser printer may hold two or three bins, depending on the model. In all bins of a laser printer, the paper is the same size and fed lengthwise. Portrait and landscape printing are controlled by the font rather than by the physical orientation of the paper.

In the case of continuous forms and twin sheet feeders, there is one form length associated with any given width; thus, specifying the forms length serves to define both dimensions of the paper. When the printer receives continuous forms or sheets from twin sheet feeders, the number of DAFVU record bytes must be twice the number of lines per form (at the vertical pitch specified by the vertical pitch record).

In the case of twin bins, however, there may be more than one length associated with a width. Twin bins and multiple bins support fractional paper lengths. For these reasons, the length of DAVFU records associated with twin and multiple bins is not strictly derived from physical paper length. Instead, the correct number of DAVFU record bytes is twice the value of a code that has been assigned to a particular paper size. Tables of form length codes are found in user manuals for printers supporting twin and multiple bins.

Table 11-7 lists form length codes assigned to two combinations of paper sizes supported by twin bin feeders. This partial listing is provided to illustrate the selection of form length codes. For a complete listing, refer to the appropriate user manual.

Table 11-7. Sample Form Length Codes

Form Length Specified	Paper Size Bin #1	Printable Lines	Paper Size Bin #2	Printable Lines
31 @ 3 pitch	8 x 10.5 in.	31	8 x 10.5 in .	31
42 @ 4 pitch	"	41	"	41
63 @ 6 pitch	"	62	"	62
84 @ 8 pitch	"	82	"	82
32 @ 3 pitch	7.25 x 10.5 in	31	7.25 x 10.5 in	31
43 @ 4 pitch	"	41	"	41
64 @ 6 pitch	"	62	"	62
85 @ 8 pitch	"	82	"	82

According to Table 11-7, if both bins hold 7.25 X 10.5 inch paper, and the desired vertical pitch is 6 lpi, the DAVFU record should show 64 lines per page. For the same combination of paper sizes and a vertical pitch of 8 lpi, the DAVFU record should show 85 lines per page. If both bins hold 8 x 10.5 inch paper and the desired vertical pitch is 8 lpi, the DAVFU record should show 84 lines per page.

Because printers require a margin for handling sheets of paper, the number of printable lines per page is less than the number of lines indicated by the DAVFU record. Table 11-7, like tables in user manuals, shows both the actual and printable number of lines.

If the DAVFU record indicates a code that is not listed in a table, the next higher code listed is selected. If the indicated code is higher than any listed, the highest code in the table is used.

11.13.8 Font Selection Record

Selecting a font through the Control Data Block establishes that font as the standard font until another font is selected from the control data block. If not selected from the control data block, the standard font is that font which is designated in the font catalog @FONTCAT as the default font.

The standard font is used for printing a record when the record's fourth print control byte specifies the standard font (PCB 3, Bits 0-2 = 000) or when PCB 3 is omitted from the record.

The length of the font selection record depends on the magnitude of the font number specified by the record. Selecting a font numbered 0-255 requires a 4-byte font selection record, whose format is shown in Figure 11-10. The font number, signified in the figure by nn, occupies the fourth byte.

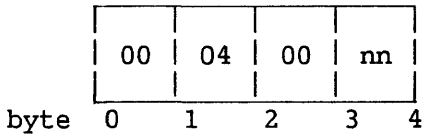


Figure 11-10. Four-Byte Font Selection Record

Selecting a font whose number exceeds 255 requires a font selection record five bytes in length, whose format is shown in Figure 11-11.

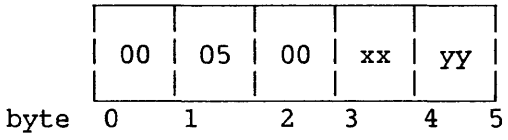


Figure 11-11. Five-Byte Font Selection Record

The maximum font number is 999 decimal; thus, the font number xxyy is a 10-bit value, and the six most significant bits of xx are ignored.

11.14 IDEOGRAPHIC PRINTING

Control of ideographic printing through print control bytes, IOCW, and IOSW will be described in a future addendum to this manual.

CHAPTER 12
WANG DISK FACILITY CHARACTERISTICS

12.1 INTRODUCTION

Table 12-1 lists the characteristics of disk drives supported by Wang VS systems. Specifications given by the table for total storage and data transfer rate assume a formatted disk.

The following drives listed in Table 12-1 can no longer be ordered from Wang Laboratories, Inc.: 2260V; 2270V series; 2280V-1,-2.

Table 12-1. Characteristics of Disk Drive Models

Disk	Q2040	2230	2260V	2265V-1
Type	Fixed	Fixed	F/R[a]	Removable
Tracks per cylinder	8	8	4	5
Cylinders	512	512	408	823
Sector size (in bytes)	256	512	256	2048
Sectors per track	32	16	24	9
Total storage (in MB)	33.55	33.55	10.03	75.85
Seek average (in ms)	65	45	38	30
Seek max (in ms)	100	80	130	55
Seek min (in ms)	10	10	9	6
Full rotation time (in ms)	20	17	25	16.66
Data transfer rate (in bytes/sec)	410K	482K	246K	1.1M

(continued)

Table 12-1. Characteristics of Disk Drive Models (continued)

Disk	2265V-2	2265V-3	2267V-1	2268V-1
Type	Removable	Fixed	Removable	Fixed
Tracks per cylinder	19	40	5	6
Cylinders	823	842	823	692
Sector size (in bytes)	2048	2048	2048	2048
Sectors per track	9	9	9	9
Total storage (in MB)	288.22	620.7	75.85	76.52
Seek average (in ms)	30	25	30	25
Seek max (in ms)	55	50	55	50
Seek min (in ms)	6	10	7	7
Full rotation time (in ms)	16.6	16.6	16.6	17.1
Data transfer rate (in bytes/sec)	1.1M	1.1M	1.1M	1.1M

Table 12-1. Characteristics of Disk Drive Models (continued)

Disk	2268V-2	2268V-4	2270V-1	2270V-2,-4
Type	Fixed	Fixed	Removable	Removable
Tracks per cylinder	8	24	1	2
Cylinders	1024	711	77	77
Sector size (in bytes)	2048	2048	256	1024
Sectors per track	9	13	16	8
Total storage (in MB)	150.9	454.3	0.3154	1.2
Seek average (in ms)	20	20	260	260
Seek max (in ms)	40	45	[b]	[b]
Seek min (in ms)	5	5	8	8
Full rotation time (in ms)	16.66	16.66	166	166
Data transfer rate (in bytes/sec)	1.1M	1.8M	25K	49K

(continued)

Table 12-1. Characteristics of Disk Drive Models (continued)

Disk	2270V-3	2270V-5,-6	2280V-1
Type	Removable	Removable	F/R[a]
Tracks per cylinder	2	2	1+1
Cylinders	77	40	823
Sector size (in bytes)	256/1024[c]	512	2048
Sectors per track	8	9	9
Total storage (in MB)	.3154/1.2[c]	0.368	30.34
Seek average (in ms)	260	463	30
Seek max (in ms)	[b]	[b]	55
Seek min (in ms)	8	40	6
Full rotation time (in ms)	166	200	16.6
Data transfer rate (in bytes/sec)	49K	23K	1.1M

Table 12-1. Characteristics of Disk Drive Models (continued)

Disk	2280V-2	2280V-3
Type	F/R[a]	F/R[a]
Tracks per cylinder	1+3	1+5
Cylinders	823	823
Sector size (in bytes)	2048	2048
Sectors per track	9	9
Total storage (in MB)	60.68	91.02
Seek average (in ms)	30	30
Seek max (in ms)	55	55
Seek min (in ms)	6	6
Full rotation time (in ms)	16.6	16.6
Data transfer rate (in bytes/sec)	1.1	1.1M

[a] Fixed/Removable
[b] Specification not supplied by manufacturer
[c] Hard-sectored/Soft-sectored

The 2265V, 2267V, and 2268V series are available as dual port drives, which can be concurrently configured in two VS systems. Dual port drives are denoted by a model number ending with "A" (i.e., 2265V-1A, 2265V-2A, 2265V-3A; 2267V-1A; 2268V-1A, 2268V-2A, 2268V-4A.).

Q2040 is not a model number; it is the designation used by the GENEDIT utility for the fixed disk drive internal to the VS25 and VS45 systems.

The following information is provided to aid in distinguishing between models in the 2270V series.

Models 2270V-1, -2, -3, and -4 support 8-inch diskettes. Of these, 2270V-1, -2 and -3 are used in archiving workstations; 2270V-4 is an internal drive for VS25 and VS45 systems. Of the archiving workstation units, 2270V-1 supports hard-sectored diskettes, 2270V-2 supports soft-sectored diskettes, and 2270V-3 supports both hard- and soft-sectored diskettes.

Models 2270V-5 and 2270V-6 support 5.25-inch diskettes. Of these, 2270V-5 is an internal drive for VS15 and VS65 systems, and 2270V-6 is used in an archiving workstation.

12.2 LOGICAL AND PHYSICAL SECTORS

A physical sector is the smallest addressable unit of data on a disk. A logical sector is the smallest unit of data transferred between disk and main memory. Table 12-1 shows the physical sector size of VS disk drives. The logical sector size for all disks is 2048 bytes. Sector sizes, both physical and logical, do not include control information (i.e., headers and ECC bytes).

12.3 DISK DRIVES AND I/O PROCESSORS

Table 12-2 lists, by class of VS system and central processor (CP) type, the supported combinations of I/O processors and VS disk drives. The term "I/O processors" refers to Device Adapters (DAs), I/O Processors (IOPs), and I/O Controllers (IOCs).

Of the I/O processors listed in Table 12-2, only the following can be currently ordered from Wang Laboratories, Inc.: 25V50, 22V88, and 23V98.

Table 12-2. Combinations of IOPs and Disk Drives

DA/IOP/IOC	Disk Drive	
	VS15 (CP5) VS65 (CP7)	
22V50-0	2268V-1,-2	
25V50	2265V-1,-1A-2,-2A,-3,-3A 2268V-1,-1A,-2,-2A	2267V-1,-1A 2280V-1,-2,-3
25V51	2230	
25V55	Q2040	
	VS100 (CP4)	
22V28	2265V-1,-2	2280V-1,-2,-3
22V28A	2265V-1	2280V-1,-2,-3
22V48	2265-1,-1A,-2,-2A	2280V-1,-2,-3
22V88	2265V-1,-1A,-2,-2A,-3,-3A 2268V-1,-1A,-2,-2A,-4,-4A	2267V-1,-1A 2280V-1,-2,-3
	VS300 (CP8)	
23V98	2265V-1,-1A,-2,-2A-3,-3A 2268V-1,-1A,-2,-2A,-4,-4A	2267V-1,-1A

12.4 DISK IOCW

A general description of the IOCW is found in Chapter 9. The disk I/O command word consists of a command, a memory address, a data count, and sector address, as illustrated by Figure 12-1.

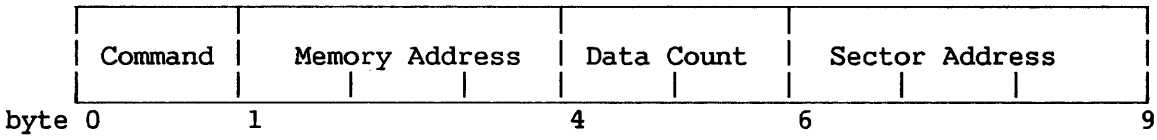


Figure 12-1. Disk IOCW

12.4.1 Command Byte

The first byte of the IOCW contains the I/O command and the command modifier bits.

The two dual port commands (Release and Reserve) are specified by Bits 0-3 and are modified by Bits 4-7. The remaining disk commands are specified by Bits 0-2 and are modified by Bits 3-7.

The following command codes are defined:

Command Code	Function
0001	Release (dual port)
0010	Reserve (dual port)
010	Read disk sector(s)
100	Write disk sector(s)
101	Write Verify
110	Seek
111	Format

The command modifier bits are defined as follows:

Command Modifier Bits	Function
10000	Release
1000	Read/Write Diagnostic
0100	Suppress Retry
0010	Indirect Addressing
0001	Removable Platter

The 5-bit command modifier Release is used only with the 3-bit commands; thus, the total of command bits and command modifier bits never exceeds eight.

12.4.2 Memory Address

If IOCW Bit 6 = 0, the IOCW directly addresses a main memory area from which or into which data is to be transferred. This area of memory must be 2048-byte aligned.

If Bit 6 = 1, the IOCW addresses the start of an indirect address list, comprised of physical addresses of data areas. The memory data area addressed by each entry of the Indirect Address list must be 2048-byte aligned, or the command will be rejected with an indication of "invalid command."

12.4.3 Data Count

For data transfer operations, the data count field of the IOCW specifies the number of bytes to be transferred between the disk and memory. This number must be divisible by the logical sector size (2048), or the command will be rejected with an indication of "invalid data count."

12.4.4 Sector Address

Bytes 6-8 of the IOCW indicate the address of the starting sector to be transferred. The value supplied in this field is interpreted as the relative number, starting at zero, of a 256-byte sector. Thus, the specified address of a 2048-byte sector is its relative sector number multiplied by eight; and the three least significant bits of the specified address must be zeros.

If the starting sector and data count imply a cylinder change, the command will be rejected with an indication of invalid command and invalid data count.

12.5 DUAL PORT COMMANDS

12.5.1 Release Command

Execution of the Release command causes the controller to give up reservation of a dual port disk. A Release command issued to a disk that is already reserved by the other controller or is not dual ported has no effect; the controller returns a Normal Completion IOSW.

12.5.2 Reserve Command

Execution of the Reserve command causes the controller to attempt reservation of a dual port disk. If the disk is already reserved by another controller, the controller returns an IOSW with Error Completion and Disk Unavailable bits set. Otherwise, the returned IOSW indicates normal completion. A Reserve command issued to a disk that is not dual ported has no effect; the controller returns a Normal Completion IOSW.

12.6. I/O COMMANDS

12.6.1 Read Command

Execution of a Read command positions the access mechanism and transfers information from the disk into memory. The access mechanism is positioned at the specified sector within the specified cylinder. The disk verifies (using the Error Correction Code attached to each sector) that the data as read is valid. Also, it verifies that the sector(s) read are those requested, by comparing the sector address identification with the specified sector of the Read command.

If any error is detected and automatic retries are enabled (IOCW Bit 5 = 0), the I/O device initiates the appropriate retry procedures. If retry is successful, normal processing is continued. In this case, the error is reported at the end of the I/O sequence by means of an IOSW with both NC and EC bits set. The appropriate error status bits are set in the IOSW to describe the error.

If an irrecoverable error occurs (i.e., if all retry attempts fail), then the Read operation terminates and the error is reported as irrecoverable by means of an Error Completion IOSW. The residual byte count indicates the amount of data transferred before termination.

The Read command allows any number of sectors on a cylinder to be read by one command.

12.6.2 Write and Write (Verify) Commands

Execution of a Write command positions the access mechanism and transfers information from memory to the disk. The access mechanism is positioned to the specified cylinder and sector within the cylinder. As the Write operation is performed, an Error Correction Code is computed by the disk controller and appended to the sector record.

The Write Verify command requires a second revolution of the disk mechanism. The contents of the sectors as written on the disk are validated by a reading of all the data written and a comparison of that data with the data as it is found in memory. As part of the Read operation, the ECC is recalculated and checked.

If an error is found during data verification, the IOSW data comparison error bit is set. If automatic retries are enabled (IOCW Bit 5 = 0), the I/O device initiates the appropriate retry procedures. If all retry attempts fail, the operation is terminated and an Error Completion IOSW is issued with a nonzero residual byte count.

For disk drives configurable to the VS100 (listed in Table 12-2) and for the 2230 drive, the stored residual byte count indicates the sector in error. This residual count is decremented each time a sector is successfully verified. Thus, an uncorrectable error on the first sector written would result in a residual count equal to the original data count; an error on the second sector written would result in a residual count equal to the original count minus 2048, and so forth.

If retry is successful, the operation is reported back through the IOSW with both Normal Completion (NC) and Error Completion (EC) bits set.

The Write command allows any number of sectors on a cylinder to be written.

12.7 DISK CONTROL COMMANDS

12.7.1 Seek

Execution of a Seek disk address command positions the disk access mechanism. The data count field and the memory address field of the IOCW are ignored. No validity checking of the actual disk mechanism position occurs during this command. The Seek operation is automatic on Read, Write, and Format commands.

12.7.2 Format

The Format command causes the addressed sectors to be formatted with sector preambles. The data in the sectors after successful execution of the command is unpredictable.

The data count of a Format IOCW can be any number that is a multiple of the logical sector size (X'800'), less than X'10,000', and does not imply a cylinder change.

12.8 COMMAND MODIFICATION

IOCW Bits 4-7 modify dual port commands; Bits 3-7 modify other commands.

12.8.1 Release

When Bit 3 is set to 1, the operation specified by Bits 0-2 is performed; then the controller releases the disk. The other controller may then reserve the disk and perform an I/O or control operation. Bit 3 is not interpreted as a command modifier by disks that do not have dual ports.

Because a Reserve operation is implicitly performed by an I/O or control operation, a command modifier bit need not be defined for reserving a dual port disk.

12.8.2 Read/Write Diagnostic

When Bit 4 is set to 1, data transfer is limited to one physical sector. Indirect data addressing is not allowed. Specification of multiple sectors or indirect data addressing causes the return of an IOSW with EC and IC bits set; no data is transferred.

When this bit is set on a Read command, the bytes of ECC code associated with the specified sector are transferred into memory immediately following the data. ECC error recovery is suppressed. A physical sector of 256, 512, or 1024 bytes is followed by an ECC code of four bytes. A sector of 2048 bytes is followed by an ECC code of eight bytes.

When this bit is set with a Write command, the number of bytes in a physical sector, starting from the address specified in the IOCW, are transferred to the disk sector as data, as in a normal Write command. However, instead of generating an ECC code for the sector, the disk hardware uses the next four or eight bytes of memory as its ECC code and writes these into the sector. ECC error recovery is suppressed.

12.8.3 Suppress Retry

When IOCW Bit 5 = 1, any automatic retry procedures normally initiated by the device are bypassed. All error conditions are reported immediately as irrecoverable errors (IOSW bit EC set, bit NC not set).

12.8.4 Indirect Data Addressing

When IOCW Bit 6 is set to 1, the data address portion of the IOCW addresses an Indirect Address list as described in Chapter 9. The address contained in each of the Indirect Address list must be a multiple of the disk drive's logical sector size (i.e., 2048).

12.8.5 Removable Platter

When IOCW Bit 7 is set to 1, the I/O command disk address refers to the removable platter. Otherwise the command is for the fixed platter. This bit is ignored by disk drives that do not support fixed/removable platter combinations.

12.9 DISK I/O STATUS WORD

A general discussion of the I/O Status Word (IOSW) is found in Chapter 9. Figure 12-2 shows the IOSW for disks.

	General status	Error status	Extended status	Residual byte count	retry indicator	(VS300) Extended MPE/MAE
bits	0	8	16	32	48	56 63
byte	0	1	2	4	6	7 8

Figure 12-2. Disk IOSW

12.9.1 General Status Byte

IOSW Bit	Mnemonic	Meaning
0	IRQ	Set only when an "intervention required" condition is detected at the start of an operation. Not set when "Not Ready During Operation" device status is set. EC is always set when IRQ is set.
1	NC	Set on successful completion, with or without retries.
2	EC	Set when any error status bit is set, even if condition is corrected by retry. NC and EC both set if a retry is successful.
3	U	Set whenever device becomes ready after the START button at the device is pressed. NC, EC, or PC may also be set.
4	PC	IOP now ready; may be set alone or with NC, EC, or U. (Used only on the VS100 system.)
5-7		Reserved (always 0).

12.9.2 Error Status Byte

IOSW Bit	Mnemonic	Meaning
8	IC	Set to indicate invalid command byte in IOCW; IOCW not fullword aligned; Indirect Address list not fullword aligned; data area(s) not 2048-byte aligned. EC always set if IC is set.
9	MPE	Set on occurrence of main memory parity error during transfer of Command Table Address (CTA), IOCW, Indirect Address list, or data. EC always set if MPE is set. On VS300 only, MPE is qualified by IOSW Byte 7.
10	MAE	Set on occurrence of main memory addressing error during transfer of IOCW, Indirect Address list, or data. EC always set if MAE is set. On VS300 only, MPE is qualified by IOSW Byte 7.

IOSW Bit	Mnemonic	Meaning
11	DM	Set only if one or more of the following conditions is indicated in the device-dependent status bytes: • Sector overrun • Seek incomplete • Not ready during operation • Timeout on sector • Data compare error • Invalid sector ID • Invalid CRC or ECC check • Overrun
12	DAM	Set whenever command terminated with error (EC set; NC not set) after main memory or data on disk has been modified.
13	IL	Never set.
14-15		Reserved (always 0).

12.9.3 Extended Status Bytes

The Extended Status bytes consist of 16 bits that indicate specific disk and/or controller states.

IOSW Bit	Mnemonic	Meaning when Set to 1
16	HDF	Sector header reformatted on Write retry
17	HDS	Sector header skipped on Read retry
18	DIA	ECC transferred (not applicable to diskette drives)
19	OPT	Optimization used
20	IDA	Invalid disk address
21	IDC	Invalid data count
22	SO	Sector overrun
23	SI	Seek incomplete
24	WP	Write protect
25	NRO	Not ready during operation
26	ST	Timeout on sector

IOSW Bit	Mnemonic	Meaning
27	DC	Data compare error
28	IID	Invalid sector ID
29	CRC	Invalid CRC or ECC
30	O	Overrun
31	DU	Disk Unavailable

Optimization Used

This bit is set to 1 whenever transfer of multiple sectors starts with a sector closer to the disk head than the sector specified by the IOCW sector address. The specified sector is transferred at a later point in the rotational cycle to take advantage of the disk's current rotational position.

Invalid Disk Address

This bit is set to 1 when the starting sector address points to a nonexistent cylinder, a condition that also causes the Error Completion bit and the Invalid Command bit to be set to 1.

Invalid Data Count

This bit is set to 1 for the following conditions:

- Data count is not a multiple of the logical sector size.
- Data count is greater than or equal to X'10,000'.
- Data count implies a cylinder change.
- Data count is 0.

These conditions cause the Error Completion bit and the Invalid Command bit to be set to 1.

Sector Overrun

This bit is set to 1 when a sector in a multiple sector operation requires an additional rotation for transfer to or from main memory. The operation is retried starting at the "missed" sector. The condition may be caused by insufficient intersector gap time due to a defective disk pack or incorrect disk drive alignment.

Seek Incomplete

This bit is set to 1 to indicate that the disk was unable to complete a Seek due to a malfunction or that a hardware fault occurred during a transfer operation.

Write Protect

This bit is set to 1 on all commands when the disk or diskette is write protected. The issuance of a Write command to a write-protected drive causes the return of an IOSW with EC, IC, and WP bits set to 1; no data is transferred.

Not Ready During Operation

This bit, along with the DAM bit, is set to 1 whenever a disk becomes not ready during an operation.

Timeout on Sector

This bit is set to 1 whenever a sector operation takes longer to complete than a sector's time. Device malfunction is indicated and the command is not retried.

Data Compare Error

This bit is set to 1 whenever the controller detects a data mismatch during the verify part of a Write Verify command.

Invalid Sector ID

This bit is set to 1, if at the start of an operation, the controller finds that the sector identifier bytes are not as expected.

Invalid CRC or ECC Check

This bit is set during a Read operation if the CRC or ECC calculated during the operation is not the same as the CRC or ECC written on the disk.

Overrun

This bit is set if memory service is not provided fast enough to keep up with the disk data transfer (rotation) speed during a Read or Write operation. Data transfer stops as soon as this condition is detected.

Disk Unavailable

This bit is set to 1 when a controller attempts an operation to a disk that is reserved by another controller. The EC bit is also set.

12.9.4 Residual Byte Count

In an error completion IOSW, this field of the IOSW specifies the number of any data bytes transferred before the I/O operation terminated unsuccessfully. If no data bytes were successfully transferred, this field shows the data count. On certain drives, mentioned in Section 12.6.2, the residual byte count identifies the sector in which a Write Verify operation encountered an uncorrectable error.

12.9.5 Retry Indicator Byte

IOSW Bit	Value	Meaning
48-51	0	No special adjustment
	1	Data strobe early adjustment applied during retry
	2	Data strobe late adjustment applied during retry
	3	Servo offset minus adjustment applied during retry
	4	Servo offset plus adjustment applied during retry
	5	Data strobe early and servo offset minus adjustments applied during retry
	6	Data strobe late and servo offset minus adjustments applied during retry
	7	Data strobe early and servo offset plus adjustments applied during retry
	8	Data strobe late and servo offset plus adjustments applied during retry
	9	ECC applied on Read retry
52-55	n	Number of retries (n) for the above adjustments before terminating

Some disks do not support all of the adjustments listed above.

When invalid data is detected on a Read or Write Verify operation, retries of a certain type (indicated by Bits 48-51) are performed. After each retry, the retry count (Bits 52-55) is incremented. After the maximum number of retries, the retry indicators (Bits 48-51) are updated to indicate the type of retry attempted, the retry count is zeroed, and the next type of retry is attempted.

For the 2270V-1, the retry indicators (Bits 48-55) are updated after a maximum of 16 retries, with no special adjustment, for Read and Write operations.

For the 2230, the drive attempts to read an invalid logical sector four times. If the fourth attempt is unsuccessful, the drive attempts to read each of the four physical sectors comprising the logical sector. After eight unsuccessful attempts to read a single sector, the drive attempts ECC, setting the retry indicator to a value of 9.

12.9.6 IOSW Byte 7

On the VS300, Byte 7 of an error completion IOSW can contain extended status information on memory address and memory parity errors; Byte 7 of a normal completion IOSW can contain diskette drive indicators.

On systems other than the VS300, IOSW Byte 7 is used only for diskette drive indicators.

Extended MPE/MAE Byte (VS300 Only)

When the EC and MPE bits are both set to 1, IOSW Byte 7 shows one of the following hexadecimal codes:

Code	Meaning
01	System memory data error
04	System bus memory read parity error
08	System bus parity error

When the EC and MAE bits are both set to 1, IOSW Byte 7 shows one of the following hexadecimal codes:

Code	Meaning
02	Illegal system memory address
10	Illegal system memory page access
20	Illegal I/O command from IOC

Diskette Drive Indicator Bits

IOSW Bit	Meaning when Set to 1
56	Diskette write protected
60	Soft-sectored medium
61	Double-sided medium (valid only for soft-sectored diskettes)

12.9.7 Disk Unsolicited Interruptions

When a disk drive first becomes ready (i.e., after a disk platter is mounted), an unsolicited "device now readied" interruption request is generated.

NOTE

Disk I/O Processors do not support the HALT I/O machine instruction. If this command is received, a condition code of 0 is returned if the specified device is idle. If an operation is in progress or an interruption is pending, a condition code of 1 is returned and disk processing continues until the operation completes.

CHAPTER 13
WANG MAGNETIC TAPE CHARACTERISTICS

13.1 INTRODUCTION

All Wang systems support magnetic tape drives. The VS15, VS45, and VS65 systems support only serial tape drives -- Wang models 2509V and 2529V. Other VS systems also support parallel tape drives -- the 2209V and 2219V series of Wang models.

All tape drives except the 2529V are reel-to-reel, using 0.5-inch (1.27-cm) tape. The 2529V is a cartridge drive, using 0.25-inch (0.64-cm) tape. The functional characteristics of reel-to-reel tape drives, whether parallel or serial, are basically similar; there are major functional differences, however, between reel-to-reel tape drives and the cartridge tape drive.

All tape drives and their controllers are compatible with industry standards and are designed to facilitate information interchange between the Wang VS and other computer systems.

Table 13-1 summarizes the characteristics of VS tape drives. The speeds listed in the last column apply to Read and Write operations rather than to control operations such as Rewind.

Table 13-1. VS Tape Drive Characteristics

Model	Tracks	Recording Density (bpi) /Mode	Speed (ips)
Parallel Tape Drives			
2209V-1	9	1600/PE	75
2209V-2	9	800/NRZI, 1600/PE	75
2209V-3	7	800/NRZI	75
2219V-1	9	1600/PE, 6250/GCR	75
2219V-2	9	1600/PE, 6250/GCR	125
2219V-3	9	800/NRZI, 1600/PE, 6250/GCR	75
2219V-4	9	800/NRZI, 1600/PE, 6250/GCR	125
Serial Tape Drives			
2509V	9	1600/PE	75
2529V	4	6400	30

13.2 GENERAL DESCRIPTION OF REEL-TO-REEL TAPE DRIVES

Reel-to-Reel tape drives comprise all tape drives except the 2529V cartridge tape drive.

13.2.1 Track Allocation

Information is written on tape by magnetizing small discrete positions across the width of the tape. The result is a column of bits that represent a byte of information plus parity. Bit positions do not correspond sequentially to track numbers across the tape. Bit positions are allocated as shown in Figure 13-1. The parity bit is indicated by the symbol "P" in the figure. Tracks are numbered from the near edge with the oxide side down and with the take-up reel on the right.

	9-Track	7-Track
Track number	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7
Bit position	4 6 0 1 2 P 3 7 5	P 0 1 2 3 4 5

Figure 13-1. Tape Bit Positions

Bit positions 0-5 on the 7-track tape correspond to Locations 2-7 in main memory.

13.2.2 Tape Markers

Magnetic tape must have some blank space at the beginning and end of the reel to allow threading it through the feed mechanism of the tape unit. Markers called reflective strips are placed on the tape by the operator to enable the tape unit to sense the beginning and the end of the usable portion of the tape. The tape unit senses a marker either as the load point marker, where reading or writing is to begin, or as the end-of-tape marker, approximately where writing is to stop.

Load Point Marker

At least 10 feet (3 meters) of tape must be allowed between the beginning of the reel and the load point marker as a leader for threading the tape on the tape unit. To indicate the load point, the marker must be parallel to and not more than 0.31 inch (0.08 cm) from the edge of the tape nearest the tape unit.

End-of-Tape Marker

About 14 feet (4.3 meters) of tape are usually reserved between the end-of-tape marker and the end of the tape. This space includes at least 10 feet (3 meters) of leader and 4 feet (1 meter) for the recording of data after the end-of-tape marker is sensed. When the tape is mounted, the marker is placed parallel to and not more than 0.31 inch (0.08 cm) from the edge of the tape nearest the tape unit. The end-of-tape reflective marker indicates the beginning of the end-of-tape area.

13.2.3 Tape Mark

The tape mark is a special single-byte block written only by the Write Tape Mark command. The value written is X'13' (X'0F' for 7-track tapes). It can be used to separate multiple files (or any other collection of blocks) on a tape.

13.2.4 Tape Blocks

Information on tape is arranged in blocks, as shown in Figure 13-2. A tape block usually consists of a single record. Blocks are separated on tape by an interblock gap -- a length of blank tape of approximately 0.6 inch (1.52 cm). During writing, the gap is always produced at the end of a block. A tape block is therefore defined or marked by an interblock gap before and after the block. Maximum block length is 32,768 bytes.

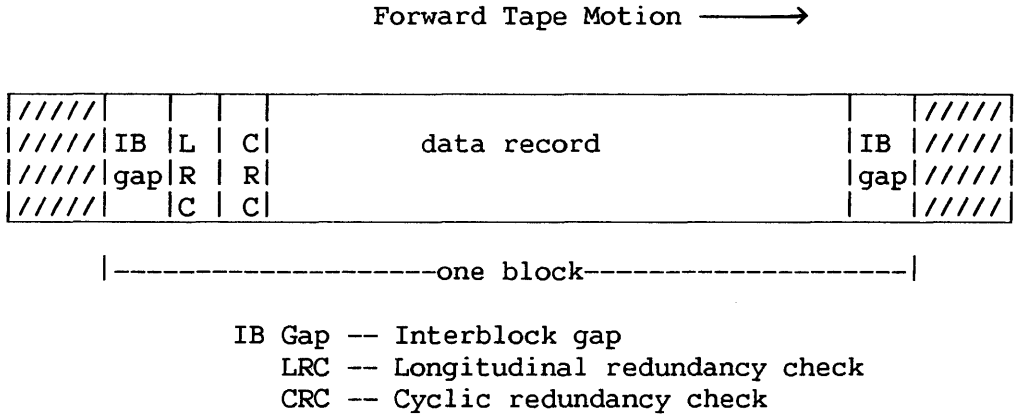


Figure 13-2. Reel-to-Reel Tape Blocks

13.2.5 File Protection

The write operation automatically erases any previous information on the tape; therefore, a file protection device is provided to prevent accidental erasure. A plastic write-enable ring fits in a circular groove molded in the back (machine side) of the tape reel. This ring must be in place for the machine to write on the tape in the reel. When the ring is removed, only reading can take place; the tape is thus protected from accidental writing, which could erase valuable information.

13.2.6 Checking Tape Validity

The validity of information written on or read from tape is ensured through the use of longitudinal, vertical, and skew checks; there is also a cyclic redundancy check for 9-track tape. The checking of tape information is accomplished during a Read operation or during a read check of a Write operation.

13.3 GENERAL DESCRIPTION OF THE CARTRIDGE TAPE DRIVE

The 2529V tape drive supports 0.25-inch (0.09-cm) tape. The tape is wound on two reels within a cartridge that is inserted into the drive.

13.3.1. Track Allocation

Four tracks run lengthwise along the tape, one below the other. After data has been read or written to the end of one track, the direction of tape movement is reversed and the next lower track is accessed to continue the operation.

Recorded data forms a single row of bits along the length of the track. For every eight bits of data there is a parity bit.

13.3.2 Tape Markers

Tape markers are small perforations that demarcate the recordable area of each track from that section of track used for threading the tape onto reels. A tape marker on Track 0, the load point (LP) marker, indicates to the tape drive where reading or writing is to begin. A tape marker on Track 3, the End of Tape (EOT) marker, indicates to the drive approximately where writing is to stop.

13.3.3 Tape Mark

The tape mark is a special block written only by the Write Tape Mark command. It can be used to separate multiple files (or any other collection of blocks) on a tape. During Read operations, tape marks are not transferred to main memory.

13.3.4 Tape Blocks

Information on tape is arranged in blocks, as shown in Figure 13-3. A tape block usually consists of a single record. Blocks are separated on tape by an interblock gap of blank tape. During writing, the gap is always produced at the end of a block. A tape block is therefore defined or marked by an interblock gap before and after the block.

Block length must range between 2 bytes and 17K bytes, inclusive.

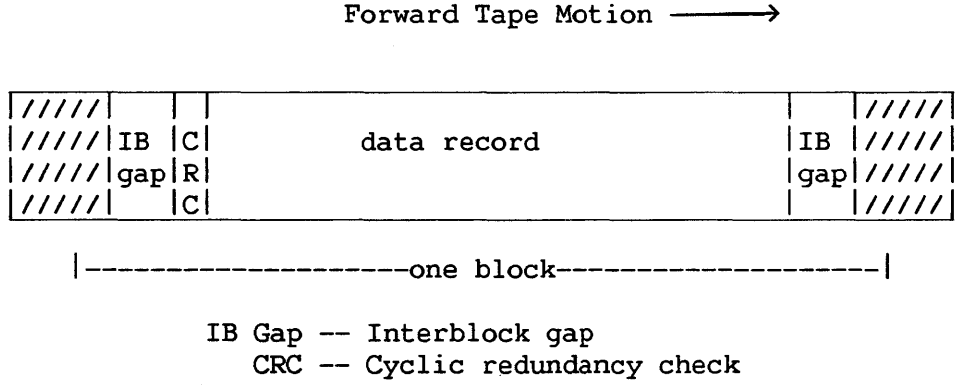


Figure 13-3. Cartridge Tape Blocks

13.3.5 File Protection

The Write operation automatically erases any previous information on the tape; therefore, a tape protection device is provided to prevent accidental erasure. A plastic knob is found on the edge of the cartridge inserted into the drive. When this knob is rotated to the SAFE position, only reading can take place; the tape is thus protected from accidental writing, which could erase valuable information.

13.3.6 Dismounting Cartridge

The cartridge is not locked in the drive during any of the drive's operations. Care must be taken not to remove the cartridge while the TAPE LOADED indicator is lit. When the cartridge is dismounted, the tape drive goes off-line and the TAPE LOADED indicator light turns off.

3.3.7 Loading Tape

There is no LOAD button on the tape drive unit. The tape is loaded (i.e., positioned to tape load marker) by the drive after the operator inserts the cartridge and presses the ONLINE button.

13.3.8 Tape Length and Thickness

The following lengths of cartridge tape are supported by the Find Tape Length command: 300 feet (91.5 meters), 450 feet (120 meters), and 600 feet (240 meters). The 300- and 450-foot tapes are of the same thickness, and thicker than the 600-foot tape.

13.3.9 High and Low Current

The electrical current passing through the drive's head on a Write operation must be adjusted to the thickness of the tape being used. The longest tape (600 feet) requires a higher current than the shorter tapes. To adjust the current, three control commands are provided through the I/O command word (IOCW): Find Tape Length, Set Write Current High, Set Write Current Low. Invoking the Find Tape Length command causes the tape length to be indicated in the I/O status word (IOSW). The current can then be set high or low accordingly.

Programming Note

The drive responds to the Find Tape Length command by rewinding the entire length of the tape. In the case of of the 450-foot (120 meter) tape, this operation takes about three minutes. The command need be invoked only once if the tape length (determined by this single invocation) is written to a header block. Subsequently, the tape length can be determined by the much briefer operation of reading the header block.

13.3.10 Checking Tape Validity

The validity of information written on or read from tape is tested through the use of parity and cyclic redundancy checks. The checking of tape information is accomplished during a Read operation or during a read check of a Write operation.

13.3.11 Auto-Retry

The tape drive can be requested to initiate and carry out a retry operation whenever invalid information is detected on a Read or Write operation. This facility, called auto-retry, is automatically enabled at Load time. It can be disabled and re-enabled by means of the IOCW control command Toggle Retry, described in Section 13.6.16. IOSW Bit 30 indicates the current state (enabled, disabled) of the facility.

Auto Retry Enabled

In order to retry Write operations, the tape drive prefixes a 2-byte block number to each block before it is written. When the block is successfully written, the drive stores in a queue the block number and number of bytes written. (The queue holds these counts only for the four most recent writes.) When a Write error is encountered, the drive uses these counts to reposition the tape to the end of the last valid block. After repositioning the tape and erasing invalid data, the drive retries the Write. After twenty unsuccessful retries, the drive returns an IOSW showing error completion, CRC error, and number of retries.

Retries of Read operations are accomplished without the appendage of block numbers. The maximum number of Read retries is thirty.

When auto retry is enabled and a Read operation is performed, the drive strips off the first two bytes of each block being transmitted to main memory. If the block was written when auto retry was enabled, these bytes contain a block count. Note that the drive does not check the mode in which the tape was written.

Auto Retry Disabled

When auto retry is disabled, no block counts are appended to data. On encountering a Read or Write error, the drive returns an error completion IOSW without performing retries. It is the responsibility of the application to handle retry operations.

13.4 TAPE IOCW

A general discussion of the IOCW is found in Chapter 9. I/O commands to the magnetic tape consist of a command, a memory address, and a data count, as shown in Figure 13-4. There is no device-dependent section.

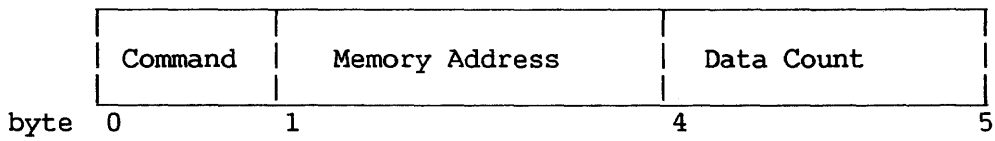


Figure 13-4. Tape IOCW

13.4.1 Command Byte

The first byte of the IOCW contains the I/O command, the command modifier bits, the indirect data addressing flag, and reduced retry flag. These are as follows:

CCMMMMIR

where CC is the command, MMMM are the command modifier bits, I is the indirect data addressing flag, and R is the reduced retry flag. There are three valid commands.

Command Code	Command Function
01	Read
10	Write
11	Control

For all drives except the 2529V, setting the reduced retry flag on a Read command reduces from 100 to 5 the number of attempts made to read a tape block before a hard error is reported. The 2529V drive ignores the reduced retry flag.

If the indirect data addressing (IDA) bit is set, the address of the memory storage from which or into which data is to be transferred will be fetched from the IDA list addressed by the memory address portion of the IOCW.

If the IDA bit is not set, the memory address portion of the IOCW directly addresses a memory area from which or into which data is to be transferred. This area of memory must be word aligned.

13.4.2 Data Count Byte

The data count portion of the IOCW contains the number of bytes to be transferred. For all drives except the cartridge drive (2529V), valid data counts range from 8 to 32,768 bytes on a Read command and from 12 to 32,768 bytes on a Write command.

For the cartridge tape drive, the minimum data count for Read and Write commands is 2 bytes. When auto retry is enabled and the command is Write, the maximum is 17K minus 2 bytes (17,406 bytes). In all other cases for the cartridge tape drive, the maximum is 17 KB (17,408 bytes).

13.5 I/O COMMANDS

13.5.1 Read

A valid Read command causes the selected tape unit to move forward to the next interblock gap. Information recorded on the tape is read and placed in contiguous ascending locations in main memory, starting with the address specified in the IOCW. If no data is detected on the tape within approximately 7 seconds after the command is issued, the operation is terminated and the incorrect length bit is set in the IOSW.

On all VS systems, reading a tape mark sets the TM indicator (Bit 28 of the IOSW). The tape mark is not sent to storage. On the VS300 system, sensing the EOT marker during a Read operation sets the EOT indicator (Bit 29 of the IOSW).

13.5.2 Write

A valid Write command causes the tape unit to move the tape forward, writing data fetched from main memory, starting with the address specified in the IOCW. The number of bytes to be transferred is specified in the data count field. These bytes are written in the form of a single physical record (a block) with vertical, longitudinal, and cyclic redundancy checks for parity applied where supported and appropriate. Sensing the EOT tape marker during a Write operation sets the EOT indicator in the IOSW.

13.6 TAPE CONTROL COMMANDS

Control operations are specified by a command code of B'11' (IOCW Bits 0-1) modified by Bits 2-5. Table 13-2 shows the control command modifier codes defined for Bits 2-5. The significance of command modifier codes B'0000' through B'1011' is common to all tape drives. The significance of the four remaining command modifier codes is drive-specific.

Table 13-2. Control Command Modifier Codes

Bits	Drive Models			
2-5	2209V-1,-2,-3	2219V-1,-2,-3,-4	2509V	2529V
0000	Sense	Sense	Sense	Sense
0100	Erase Tape	Erase Tape	Erase Tape	Erase Tape
0101	Write Tape Mark	Write Tape Mark	Write Tape Mark	Write Tape Mark
0110	Forward Space Block	Forward Space Block	Forward Space Block	Forward Space Block
0111	Forward Space File	Forward Space File	Forward Space File	Forward Space File
1000	Rewind	Rewind	Rewind	Rewind
1001	Rewind/Unload	Rewind/Unload	Rewind/Unload	Rewind/Unload
1010	Backspace Block	Backspace Block	Backspace Block	Backspace Block
1011	Backspace File	Backspace File	Backspace File	Backspace File
1100	Set Density High	Set Mode to PE	Not Used	Find Tape Length
1101	Set Density Low	Set Mode to NRZI	Not Used	Set Write Current High
1110	Set Parity Odd	Set Mode to GCR	Not Used	Set Write Current Low
1111	Set Parity Even	Drive Selected Mode	Not Used	Toggle Retry

A valid control command causes the specified control function to be executed. At the end of the control function, a normal completion IOSW is issued.

13.6.1 Sense

Execution of the Sense command senses the device's current status and sets accordingly the Load Point, Offline, File Protect, End of Tape, and Density bits.

13.6.2 Erase Tape

Execution of an Erase Tape command erases approximately 3.75 inches (9.52 cm) of tape (3 inches, or 7.62 cm, for the 2529V drive). Performing this operation in the EOT area sets the EOT indicator in the IOSW.

13.6.3 Write Tape Mark

Execution of the Write Tape Mark command writes a tape mark (a special block) on tape. Performing this operation in the EOT area sets the EOT indicator in the IOSW.

13.6.4 Forward Space Block

Execution of the Forward Space Block command moves the tape forward to the next interblock gap. Sensing a tape mark sets the TM indicator in the IOSW.

13.6.5 Forward Space File

Execution of the Forward Space File command moves the tape forward to the interblock gap beyond the next tape mark. Sensing the tape mark does not cause the TM indicator to be set in the IOSW.

13.6.6 Rewind

Execution of a Rewind control command causes the tape drive to enter rewind mode and reposition the tape at the load point tape marker. This causes the load point (LP) indicator to be set in the IOSW.

13.6.7 Rewind and Unload

2209V Series and 2509V Tape Drive

On these tape drives, execution of a Rewind/Unload control command causes the tape drive to enter rewind mode, reposition the tape at the LP tape marker, and reset the tape status to off-line.

2219V Series

On these tape drives, execution of a Rewind/Unload control command winds all of the tape onto the supply reel and reset the tape status to off-line.

2529V Tape Drive

On the cartridge tape drive, execution of the Rewind/Unload control command does not actually cause rewinding of the tape. Instead, the tape drive winds the tape forward to End-of-Tape and resets the tape status to off-line. (Subsequently, at load time, the tape is rewound to load point and retensioned in the process.)

13.6.8 Backspace Block

Execution of the Backspace Block command causes the tape drive to move the tape backward to the nearest interblock gap or to the load point, whichever comes first. Sensing a tape mark sets the TM indicator in the IOSW. Sensing the load point before the backspace operation sets the LP indicator in the IOSW with Error Completion (EC).

13.6.9 Backspace File

Execution of the Backspace File command causes the tape drive to move the tape backward to the interblock gap beyond the next tape mark or to the load point, whichever comes first. Sensing the load point sets the LP indicator in the IOSW.with Error Completion (EC). Sensing the load point before the backspace operation sets the LP indicator in the IOSW with Error Completion (EC). Sensing the tape mark does not cause the TM indicator in the IOSW to be set.

13.6.10 Set Density

Execution of the Set Density command logically switches the tape drive to the indicated density. The IOSW indicates the change in the density status bits. The change does not become effective until an I/O operation is initiated, at which point the wrong density status bit appears in the IOSW if the selected density is unavailable. This command is valid only for 9-track drives, and effective only for dual- and tri-density drives.

13.6.11 Set Parity

Execution of the Set Parity command logically switches the tape drive to the selected parity. The IOSW indicates the change in the parity status bit. This command is valid only for 7-track drives.

13.6.12 Drive Selected Mode

Density can be set manually on 2219V-x drives while they are off-line. Execution of the Drive Select Mode command selects the density that has been set manually.

13.6.13 Find Tape Length

Execution of the Find Tape Length command causes IOSW Bit 20 and/or 21 to be set, indicating a cartridge tape length of 600 feet (240 meters), 450 feet (120 meters), or 300 feet (91.5 meters). As explained in Section 13.3.9, this command is preparatory to a command that sets the write current.

13.6.14 Set Write Current High

Execution of this command sets the current that magnetizes a cartridge tape during a Write operation to the level appropriate for the thickness of a 600-foot (240-meter) cartridge tape. This command applies only to the 2529V drive.

13.6.15 Set Write Current Low

Execution of this command sets the current that magnetizes a cartridge tape during a Write operation to the level appropriate to the thickness of a 450-foot (120-meter) or 300-foot (91.5-meter) tape. This command applies only to the 2529V drive.

13.6.16 Toggle Retry

Execution of this command enables auto-retry mode if it is disabled, or disables it if it is enabled. This command applies only to the 2529V drive. The auto retry facility is explained in Section 13.3.11.

13.7 EFFECT OF TAPE MARKERS ON IOSW BITS

I/O commands and certain control commands, when encountering a tape mark or tape marker, set LP, TM or EOT bits in the IOSW. Table 13-3 shows these commands and the IOSW bit(s) that they set when encountering a tape mark or tape marker.

Table 13-3. Effect of Tape Markers on IOSW Bits

Tape Operation	Load Point Mark (LP)	Tape Mark (TM)	End-of-Tape Mark (EOT)
Write	-	-	EOT
Read	-	TM	EOT (VS300 only)
Rewind	LP	-	-
Rewind/Unload	-	-	-
Write Tape Mark	-	-	EOT
Forward Space Block	-	TM	-
Forward Space File	-	-	-
Backspace Block	LP (EC)	TM	-
Backspace File	LP (EC)	-	-

13.8 TAPE I/O STATUS WORD

A general discussion of the I/O status word (IOSW) is found in Chapter 9. Figure 13-5 shows the IOSW format for tapes.

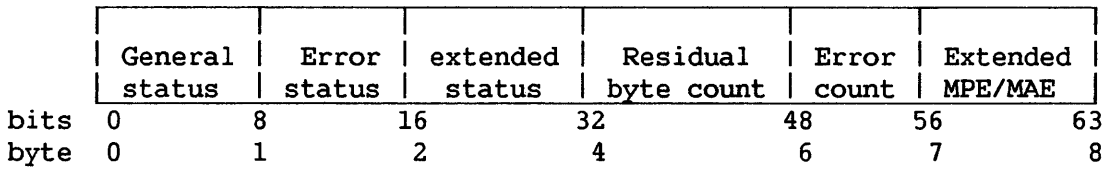


Figure 13-5. Tape IOSW Format

A full IOSW is always stored on tape I/O completion. The error status byte is stored even if the completion is not an error completion (bit EC of general status not set).

When a tape, after being mounted, first reaches load point and the tape drive is on-line, an unsolicited interruption occurs with general status bit U (unsolicited interruption) set. IOSW bits LP (load point) and FP (file protected), or both, may be set.

13.8.1 General Status Byte

IOSW Bit	Mnemonic	Meaning
0	IRQ	Never set.
1	NC	Set on successful completion with or without retries.
2	EC	Set when any error status bit is set, even if the condition has been corrected by retry. NC and EC are both set when a retry is successful.
3	U	Set whenever the device becomes ready after the ONLINE button at the device is pressed.
4	PC	IOP now ready; may be set alone or with NC, EC, or U. (Used only on the VS100 system.)
5-7		Reserved (always 0).

13.8.2 Error Status Byte -- Reel-to-Reel Tape Drives

IOSW Bit	Mnemonic	Meaning
8	IC	Set, with EC, to indicate an invalid command byte in IOCW for any of the following reasons: IOCW not fullword-aligned Indirect Address List not fullword-aligned or at an invalid address Data directly addressed by IOCW or addressed by Indirect Address List is not fullword-aligned. Attempt made to select parity on a 9-track drive Attempt made to issue diagnostic commands except on a 9-track NRZI drive Attempt made to use data length over 32,768 Attempt made to backspace when tape is at load point Attempt made to write, write tape mark, or erase tape when tape is write protected Attempt made to specify a Read operation of less than 8 bytes, or a Write operation of less than 12 bytes
9	MPE	Set in conjunction with EC on occurrence of main memory parity error during reading of CTA, IOCW, Indirect Address List, or data. On VS300 only, MPE is qualified by IOSW Byte 7.
10	MAE	Set in conjunction with EC on occurrence of main memory addressing error during reading of IOCW, Indirect Address List or data, or writing of data. On VS300 only, MAE is qualified by IOSW Byte 7.

IOSW Bit	Mnemonic	Meaning
11	DM	Set if one or more of the following conditions exist: Memory addressing error Memory parity error Memory overrun CRC or ECC error LRC error VRC error Parity error Tape drive not ready during operation Tape drive off-line during operation Given density not available at tape drive
12	DAM	Set whenever command terminated with error (EC set; NC not set) after main memory or data on tape has been modified.
13	IL	Set if the size of a block read from tape is not the same as the data count in the IOCW. If the IOCW residual count is 0, the block was longer than expected; if not, it reflects the difference between the data count and the block length.
14-15		Reserved (always 0).

13.8.3 Error Status Byte -- Cartridge Tape Drive

IOSW Bit	Mnemonic	Meaning
8	IC	Set, with EC, to indicate an invalid command byte in IOCW for any of the following reasons: IOCW not fullword-aligned Indirect Address list not fullword-aligned or at an invalid address Data not properly aligned (i.e., not fullword aligned for direct addressing operation or for first entry in Indirect Address list, or not 2048-byte aligned for subsequent entries in Indirect Address list) Attempt made to select parity, recording density, and mode Attempt made to backspace when tape is at load point

IOSW Bit	Mnemonic	Meaning
8	IC	Attempt made to write, write tape mark, or erase tape when tape is write protected Attempt made to specify a Read or Write operation of less than 2 bytes Attempt made to specify a Write operation of more than 17,406 bytes or a Read operation of more than 17,408 bytes when auto retry is enabled Attempt to specify a Read or Write operation of more than 17,408 bytes when auto retry is disabled
9	MPE	Set in conjunction with EC on occurrence of main memory parity error during reading of CTA, IOCW, Indirect Address List, or data. On VS300 only, MPE is qualified by IOSW Byte 7.
10	MAE	Set in conjunction with EC on occurrence of main memory addressing error during reading of IOCW, Indirect Address List or data, or writing of data. On VS300 only, MAE is qualified by IOSW Byte 7.
11	DM	Set if one or more of the following conditions exist: Memory addressing error Memory parity error Memory overrun CRC or ECC error Parity error Tape drive not ready during operation Tape drive off-line during operation
12	DAM	Set whenever command terminated with error (EC set; NC not set) after main memory or data on tape has been modified.
13	IL	Set if the size of a block read from tape is not the same as the data count in the IOCW. If the IOCW residual count is 0, the block was longer than expected; if not, it reflects the difference between the data count and the block length.
14-15		Reserved (always 0).

13.8.4 Extended Status Bytes -- Reel-to-Reel Tape Drives

The extended status portion of the IOSW provides a number of bits to reflect the status of each I/O operation. This extended status consists of both the unusual conditions detected in the I/O operation and the status of the device. Unless noted otherwise, the bit definitions listed below apply to all models of reel-to-reel tape drives. These bits are discussed in detail in the following sections.

IOSW Bit	Mnemonic	Meaning
16	VRC	1 = Invalid vertical redundancy check
17	LRC	1 = Invalid longitudinal redundancy check
18	CRC	1 = Invalid cyclic redundancy check
19	SC/NB	1 = Skew check on Write/noise block on Read
20	WD	1 = Wrong density (density not available)
21	PE	1 = Phase-encoded ID burst detected
22	TR7	1 = 7-track tape drive Always 0 for 2219V-x tape drives
23	PAR	0 = Odd parity; 1 = Even parity Always 0 for 2219V-x tape drives
24	FP	1 = File protected
25-26	Density	2209V-x Tape Drives only: 00 = 1600 bpi (PE) 11 = 800 bpi (NRZI) 2219V-x tape drives only: 00 = 1600 bpi (PE) 01 = 800 bpi (NRZI) 10 = 6250 bpi (GCR)
27	LP	1 = Load point encountered
28	TM	1 = Tape mark encountered
29	EOT	1 = End-of-Tape marker encountered
30	O	1 = Overrun
31	OFF	1 = Tape off-line

Invalid Vertical Redundancy Check (VRC)

This bit is set during a Read operation if the VRC stored for a byte does not give that byte odd parity on the tape.

Invalid Longitudinal Redundancy Check (LRC)

This bit is set during a Read operation if the number of 1 bits in a track for the tape block including the LRC bit is not even.

Invalid Cyclic Redundancy Check (CRC)

This bit is set during a Read operation if the CRC character calculated during the operation is not the same as the CRC character written after the block on the tape.

Skew Check/Noise Block

This bit is set during a Write operation if excessive skew is detected. This bit is set during a Read operation if a data error is found in a block with less than 12 bytes of data

Wrong Density

This bit indicates that an I/O operation has been attempted at a density not supported by the tape formatter.

Phase-Encoded ID Burst Detected

This bit indicates that a phase-encoded ID burst has been detected during the reading of the tape from load point. It is set with bits indicating the current density of the drive.

7-Track Tape Drive

This bit indicates that the tape drive is a 7-track drive.

Parity

This bit is set when even parity has been selected.

File Protected

This bit is set whenever there is no "write-enable" ring on the tape. Absence of this ring prevents accidental writing on the tape.

Density

These bits indicate the density at which the tape drive has been set.

Load Point

This bit is set whenever the tape is at load point when the operation is completed.

Tape Mark Indicator

This bit is set whenever a tape mark has been read during a Read, Forward Space Block, or Backspace Block Operation.

End-of-Tape Indicator

On all VS systems, this bit is set when the EOT reflective strip has been detected during a Write or Write Tape Mark operation; this bit is reset when the EOT reflective strip has been detected during a Backspace operation. On the VS300, this bit is set additionally when the EOT reflective strip has been detected during a Read operation.

Overrun

This bit is set if memory service is not provided fast enough to keep up with the magnetic tape data transfer speed during a Read or Write operation. Data transfer stops as soon as this condition is detected.

Off-Line

This bit is set when the tape is sensed to be off-line during an operation.

13.8.5 Extended Status Bytes -- Cartridge Tape Drive

The extended status portion of the IOSW provides a number of bits to reflect the status of each I/O operation. This extended status consists of both the unusual conditions detected in the I/O operation and the status of the device.

IOSW Bit	Mnemonic	Meaning
16	PAR	1 = Parity error
17	---	not used
18	CRC	1 = Invalid cyclic redundancy check, faulty erase, or format error
19	---	not used
20,21	LEN1,LEN2	00 = No tape in drive 01 = 300 feet 10 = 450 feet 11 = 600 feet
22	TR7	1 = 7 tracks 0 = 4 tracks
23	CUR	1 = High 0 = Low (default)

IOSW Bit	Mnemonic	Meaning
24	FP	1 = File protected
25	NOT	1 = No tape in drive
26	NL	1= Tape not loaded
27	LP	1 = Load point encountered
28	TM	1 = Tape mark encountered
29	EOT	1 = End-of-Tape marker encountered
30	MODE	1 = Auto-retry enabled (default) 0 = Auto-retry disabled
31	OFF	1 = Tape off-line

These bits are discussed in detail in the following paragraphs.

Parity Error

This bit is set to indicate that a parity error has been found.

Invalid Cyclic Redundancy Check (CRC)

This bit is set during a Read operation if the CRC character calculated during the operation is not the same as the CRC character written after the block on the tape.

Tape Length

These two bits, which indicate the length of the tape, are set by execution of the Find Tape Length IOCW control command.

7-Track Tape Drive

This bit indicates that the number of tape tracks is seven or four, when set to 1 or 0 respectively. Currently, only four tracks are supported.

Current

This bit indicates the Write current in effect. The default Write current is LOW.

File Protected

This bit is set whenever the write protect knob on the tape cartridge is rotated to the SAFE position.

No Tape In Drive

This bit is set to indicate that there is no tape in the drive and that one must be inserted before the requested action can be performed.

Tape not Loaded

This bit is set to indicate that the tape in the drive is not loaded. When the tape drive encounters a fatal error, it unloads the tape, then sets this bit in the next solicited IOSW.

Load Point

This bit is set at the the completion of a successful Load operation.

Tape Mark Indicator

This bit is set whenever a tape mark has been encountered during a Read, Forward Space Block, or Backspace Block operation.

End-of-Tape Indicator

On all VS systems, this bit is set whenever the EOT tape marker has been detected during a Write or Write Tape Mark operation. On the VS300, this bit is set additionally whenever the EOT marker has been detected during a Read operation.

Retry Mode

This bit is set to indicate that the device is not in auto-retry mode and, upon detecting a read or write error, will not retry the operation before returning an error completion IOSW. When this bit is set, the application is entirely responsible for handling retry operations. A bit value of 0 indicates that the device is in auto retry mode and, after detecting a Read or Write error, will retry the operation before returning an IOSW that indicates the error.

Off-Line

This bit is set when the tape is sensed to be off-line during an operation.

13.8.6 Error Count Byte -- Reel-to-Reel Tape Drives

When an error is detected, the error count is incremented by 1 and the operation is retried.

For a Write operation, a maximum of 50 retries of the following sequence is attempted: the tape is backspaced over the block just written, an Erase operation is initiated to erase over 3.75 inches (9.5 cm) of tape, and the Write operation is retried.

For a Read operation, a maximum of 100 retries (5 retries if IOCW Bit 7 is set to 1) of the following sequence is attempted: the tape is backspaced over the block just read, and the read operation is retried.

13.8.7 Error Count Byte -- Cartridge Tape Drives

When the device is in auto retry mode and an error is detected, the error count is incremented by 1 and the operation is retried. For a Write operation, a maximum of 20 retries is performed. For a Read operation, a maximum of 30 retries is performed; the tape is retensioned during each retry after the twentieth.

13.8.8 Extended MPE/MAE Byte

On the VS300 only, IOSW Byte 7 provides extended status information on memory address and memory parity errors.

When the EC and MPE bits are both set to 1, IOSW Byte 7 shows one of the following hexadecimal codes:

Code	Meaning
01	System memory data error
04	System bus memory read parity error
08	System bus parity error

When the EC and MAE bits are both set to 1, IOSW Byte 7 shows one of the following hexadecimal codes:

Code	Meaning
02	Illegal system memory address
10	Illegal system memory page access
20	Illegal I/O command from IOC

13.8.9 Unsolicited Interrupt at Load Time

Tape drives issue an unsolicited interrupt at load time to signal readiness for commands. The actions preceding the interrupt and the associated IOSW differ for reel-to-reel and cartridge drives.

Reel-to-Reel Tape Drives

The tape drive issues an unsolicited interrupt at Step 4 of the following sequence:

1. Operator physically mounts tape.
2. Operator pushes LOAD button, causing tape to be advanced to load point.
3. Operator pushes ONLINE button.
4. Drive issues unsolicited interrupt; LP bit is set in associated IOSW.

Cartridge Tape Drive

The tape drive issues an unsolicited interrupt at Step 3 of the following sequence:

1. Operator inserts cartridge in drive.
2. Operator pushes ONLINE button.
3. Drive issues unsolicited interrupt.
4. When processing first IOCW, drive positions tape to load point, then attempts to perform requested operation.

There is no LOAD button on the cartridge tape drive.

APPENDIX A
OPERATION CODE AND ASCII CHARACTER LIST

Op Code	Mnemonic	Format	Character	Op Code	Mnemonic	Format	Character
00	---	RR		2B	SDR, SER	RR	+
01	BCS	RR		2C	MDR, MER	RR	'
02	SIO	RR		2D	DDR, DER	RR	-
03	HIO	RR		2E	CID	RR	.
04	RTC	RR		2F	CDI	RR	/
05	BALR	RR		30			0
06	BCTR	RR		31			1
07	BCR	RR		32			2
08	POP	RR		33			3
09	POPH	RR		34			4
0A	SVC	RR		35			5
0B	PUSH	RR		36			6
0C	CIO	RR		37			7
0D	SPM	RR		38			8
0E	MVCL	RR		39			9
0F	CLCL	RR		3A	AQR	RR	:
10	LPR	RR		3B	SQR	RR	;
11	LNR	RR		3C	MQR	RR	<
12	LTR	RR		3D	DQR	RR	=
13	LCR	RR		3E			>
14	NR	RR		3F			?
15	CLR	RR		40	STH	RX	@
16	OR	RR		41	LA	RX	A
17	XR	RR		42	STC	RX	B
18	LR	RR		43	IC	RX	C
19	CR	RR		44	EX	RX	D
1A	AR	RR		45	BAL	RX	E
1B	SR	RR		46	BCT	RX	F
1C	MR	RR		47	BC	RX	G
1D	DR	RR		48	LH	RX	H
1E	ALR	RR		49	CH	RX	I
1F	SLR	RR		4A	AH	RX	J
20	LPDR, LPER	RR	(Space)	4B	SH	RX	K
21	LNDR, LNER	RR	!	4C	MH	RX	L
22	LTDR, LTER	RR	"	4D	LT	RX	M
23	LCDR, LCER	RR	#	4E	CVD	RX	N
24	HDR, HER	RR	$	4F	CVB	RX	O
25	LDER, LRER	RR	%	50	ST	RX	P
26	RPC	RR	&	51	DSEM	RX	Q
27	SVCX	RR	'	52	ENQ	RX	R
28	LDR, LER	RR	(	53	ENSK	RX	S
29	CDR, CER	RR	)	54	N	RX	T
2A	ADR, AER	RR	*	55	CL	RX	U

Op Code	Mnemonic	Format	Character
56	0	RX	V
57	X	RX	W
58	L	RX	X
59	C	RX	Y
5A	A	RX	Z
5B	S	RX	[
5C	M	RX	(Backslash)
5D	D	RX	]
5E	AL	RX	(Up-Arrow)
5F	SL	RX	(Back-Arrow or underscore)
60	STD, STE	RX	'
61	JSCI	RX	a
62	LC	RX	b
63			c
64			d
65	RBCX	RRL	e
66	RBXH	RRL	f
67	RBXLE	RRL	g
68	LD, LE	RX	h
69	CD, CE	RX	i
6A	AD, AE	RX	j
6B	SD, SE	RX	k
6C	MD, ME	RX	l
6D	DD, DE	RX	m
6E	AW, AU	RX	n
6F			o
70			p
71	RLA	RL	q
72	RPUSHA	RL	r
73	RBALS	RL	s
74			t
75	RBAL	RL	u
76	RBCT	RL	v
77	RBC	RL	w
78			x
79			y
7A	AQ	RX	z
7B	SQ	RX	
7C	MQ	RX	
7D	DQ	RX	
7E	CVQ	RX	
7F	CVP	RX	
80			
81	BALS	RX	
82	LPCW	S	
83			
84	POPN	RX	
85	PUSHN	RX	

Op Code	Mnemonic	Format	Character
86	BHX	RS	
87	BXLE	RS	
88	SRL	RS	
89	SLL	RS	
8A	SRA	RS	
8B	SLA	RS	
8C	SRDL	RS	
8D	SRDL	RS	
8E	SRDA	RS	
8F	SLDA	RS	
90	STM	RS	
91	TM	SI	
92	MVI	SI	
93			
94	NI	SI	
95	CLI	SI	
96	OI	SI	
97	XI	SI	
98	LM	RS	
99	BALCI	RS	
9A			
9B	Extended opcodes (see list below)		
9C	BSET	SI	
9D	BRESET	SI	
9E	BTEST	SI	
9F	RRCB	SI	
A0	DEQ	RS	
A1	DESK	RS	
A2	ISEM	RS	
A3	LSCTL	RS	
A4	STSCTL	RS	
A5			
A6	POPM	RS	
A7			
A8	LOT	RX	
A9	PUSHM	RS	
AA			
AB	Reserved for diagnostic use		
AC	STNSM	RS	
AD	STOSM	RS	
AE			
AF			
B0	PUSHA	RX	
B1	LPA	RX	
B2			
B3			
B4			
B5			

Op Code	Mnemonic	Format	Character
B6	STCTL	RS	
B7	LCTL	RS	
B8	SCAN	RS	
B9			
BA			
BB			
BC			
BD	CLM	RS	
BE	STCM	RS	
BF	ICM	RS	
C0			
C1			
C2			
C3			
C4	PAL	SS	
C5			
C6			
C7			
C8			
C9			
CA			
CB			
CC			
CD			
CE			
CF			
D0			
D1	MVN	SS	
D2	MVC	SS	
D3	MVZ	SS	
D4	NC	SS	
D5	CLC	SS	
D6	OC	SS	
D7	XC	SS	
D8	POPC	SS	
D9	PUSHC	SS	
DA			
DB	UNPAL	SS	
DC	TR	SS	
DD	TRT	SS	
DE	ED	SS	
DF	EDMK	SS	
E0			
E1			
E2	MVPC	SSI	
E3			
E4			
E5	CLPC	SSI	

Op Code	Mnemonic	Format	Character
E6			
E7			
E8	MCOUNT	Special	
E9	ACHECK	Special	
EA			
EB			
EC			
ED			
EE			
EF			
F0	SRP	SS	
F1	MVO	SS	
F2	PACK	SS	
F3	UNPK	SS	
F4	UNPU	SS	
F5			
F6	COMP	SS	
F7	XPAND	SS	
F8	ZAP	SS	
F9	CP	SS	
FA	AP	SS	
FB	SP	SS	
FC	MP	SS	
FD	DP	SS	
FE			
FF			

Extended Opcodes

9B00	STDD	S
9B01-9B7F	Unused	S
9B80	STCPID	S
9B81-9B82	Unused	
9B83	STLCPID	S
9B84	Unused	
9B85	STRING	S

APPENDIX B
GLOSSARY

Address
: The location of a byte in main memory. See also Base Address and Displacement (Offset).

Address Translation
: Translation of virtual addresses supplied by a program to addresses in main memory. Address translation is done by reference to the local page table.

ASCII
: American National Standard Code for Information Interchange. The VS uses ASCII for its internal character code.

Base Address
: An absolute address in storage, contained in a general register. If general register 0 is used, a base address of 0 is assumed.

Bit
: A binary digit; the smallest unit of computer information, presented in binary form to correspond to the on or off state of a computer memory element.

Boundary Alignment
: The positioning of a field on an integral boundary (such that the address of the first byte of the field, as expressed in binary, has one or more low-order 0s). Boundary alignment is required for halfwords, words, and doublewords on the VS. Instructions must be on halfword boundaries.

Byte
: On the VS, a sequence of eight bits that constitutes the smallest addressable or transferable unit of information.

Change Bit
: The change bit in an entry of the Reference and Change Table (RCT) is turned on by hardware whenever the associated page in real storage is modified.

Condition Code
: A 2-bit field in the PCW that is set by certain arithmetic and logical instructions and tested by conditional branching instructions. The condition code remains unchanged in the PCW until an instruction changes it. The meaning of the code is described for each instruction in Chapter 8 of this manual.

Control Mode
: A state of the computer system in which normal program execution is halted and special facilities for diagnostics, restart, and debugging are made available.

Control Register
: Sixteen registers, holding 32 bits each, that contain a stack limit word, a stack limit address, the system clock, and various controls for trap handling.

Data Count Field
: Bits 32-47 of the IOCW for a READ or WRITE command, specifying the number of bytes to be transmitted between an IO device and storage.

Device-Dependent Status Area
: Bits 16-31 of the IO Status Word.

Displacement (Offset)
: The relative address of a byte beyond the base register address.

Doubleword
: On the VS, a sequence of eight bytes aligned on an 8-byte boundary.

Error Status Byte
: The second byte of the IOSW, where relatively device-independent error indications may be stored.

Extended Status Bits
: Bits 48-63 (bytes 6 and 7, counting from byte 0) of the IO Status Word.

Floating-Point Register
: Used to contain data that is to be manipulated in floating-point format. The VS has four floating-point registers, each 64 bits in length, and numbered 0, 2, 4, and 6.

General Register
: On the VS, holds 32 bits or one word and is used for arithmetic and logical manipulations and addressing. The VS has 16 general registers.

High-Order
: The leftmost bit or digit; in reference to bytes, the byte at the lowest main memory address.

Index Register
: The general register containing a 24-bit number used as the index in base-index-displacement address calculations. The index can be used to provide the address of an element within a list or an array.

Indirect Address List
: A list of words containing addresses which designate the main memory location of data areas for an I/O operation. An Indirect Address list is used to expedite the transfer of more than one page of data at a time.

Interrupt or Interruption
: A transfer of control effected by switching PCWs.

I/O Command Table (IOCT)
: A table of I/O command words (IOCWs) for the devices attached to an I/O Processor. A command table address (CTA) specifies the address in main memory of each IOCT.

I/O Command Word (IOCW)
: A variable-length area (1 to 8 bytes) that specifies the next command to be executed for a device. Byte 0 holds the command code. Bytes 1-3 hold a data address or beginning address of an indirect address list; bytes 4-5 hold the data count (number of bytes to be operated on or transferred); bytes 6 to end may hold additional device-dependent information.

Input/Output Processor
: On the VS, a small computing unit that handles the transfer of data between main storage and peripheral units, relieving the central processing unit of this slower function.

I/O Status Word (IOSW)
: Eight bytes of data, stored at main memory location 0, that pass information concerning the status of an I/O device to the central processing unit. Byte 0 is the general status byte. Byte 1 is the error status byte, stored if the error completion bit is set to 1 in the general status byte. Bytes 2-3 are the device-dependent bytes; bytes 4-5 hold the residual byte count. Bytes 2-6 (bits 16-55) are sometimes referred to as the extended status bits for disk and tape.

Least Significant
: The rightmost bit or digit.

Local Page Table
: In local memory, a table consisting of one byte for each page in a segment, containing the physical page number of that page when it is in main memory. There is one local page table for each region of virtual memory. These tables are used by the central processor in translating virtual memory addresses to main memory addresses.

Low-Order
The rightmost bit or digit; the byte at the lowest main memory address.

Main Memory
Real or physical memory in the CPU.

Masking
1. Use of the program mask field in the PCW to suppress or delay processing of interruptions so that only one at a time is processed and the link back to the current instruction address is saved. 2. Use of instructions to turn off a mask bit in the PCW's status field or program mask field corresponding to a program interruption for a specific state. Masking an interruption can allow other interrupt messages with lower priority to be handled first.

Monitor Area
An area of local CP memory that maintains a list of recently referenced T-RAM entries.

Most Significant
The leftmost bit or digit.

Operand
A field of an instruction that defines an address or element of data on which the instruction operates.

Operation Code (Op Code)
The field of an instruction that specifies the operation to be performed.

Page
On the VS, a 2048-byte block of virtual memory located in main or auxiliary storage, which can be transferred between the two automatically by the computer.

Page Frame
An area of main memory that has a 2048-byte boundary alignment.

Parity
The number of 1s in a unit of data, whether odd or even. Parity checks ensure that a bit has not been changed accidentally while being read.

Privileged Instruction
One that will cause a program interruption unless the user mode bit (bit 34) in the PCW has been set to 0. Some VS privileged instructions are CIO, HIO, LCTL, LPCW, LSREG, STNSM, STOSM, RRCB, SIO, STDD, and SVCX.

Process Level

A task can run on one of eight levels of privilege, called process levels. A user program typically runs at level 0, the lowest level of privilege. If that program calls a system service, the task's process level is raised during execution of that service. See also Ring Memory Protection.

Program Control Word (PCW)

A data item of 8 bytes maintained by the central processor to control the order in which instructions are executed and to maintain the status of the central processor. Byte 0 holds the interruption code, bytes 1-3 hold the address of the current instruction, bytes 4-5 are the status field for some causes of interruption, and bytes 6 and 7 are the program mask field.

Reference and Change Table (RCT)

In local memory, a table with two bits--reference and change--for each page frame of main memory.

Reference Bit

The reference bit is turned on by hardware whenever the associated page in real storage is referred to.

Region

An area of contiguous virtual addresses beginning on a 2,048-byte boundary, whose address translation is effected through a single local page table. A region may consist of more than one page, with only portions of a region being within main memory at any time.

Region Table

A table of at least one 16-byte entry called a region node. Each entry holds the following information: the range of virtual addresses in a region, the read and write access levels for all pages in that region, and the address of the page table for that region.

Register

A storage device in the central processor. See also General Register, Control Register, and Floating Point Register.

Residual Byte Count

Bits 32-47 of the IOSW, the data count in an IOCW minus the number of bytes transferred by an IO operation.

Ring Memory Protection

A scheme of memory protection based on process levels and read/write access levels. The access level for each region of a task's virtual address space is defined in the minimum read/write fields of the region node describing that region. During address translation, the value of a region's minimum read/write fields is compared with the the task's current process level. A process level equal to or greater than the minimum read/write values is required for read/write access to the physical memory locations mapped to the region. See also Process Levels.

Sector
That part of a track on a disk that can fit into a page (2K bytes).

Segment Control Register (SCR)
A privileged 32-bit register holding the address of a task's main memory page table for a segment.

Semaphore
On the VS, a doubleword data type consisting of a 1-byte count field and head and tail pointers to elements of a first-in first-out list.

Shared Subroutine Library
A file of commonly used subroutines that can be linked during runtime to all programs that contain calls to those subroutines. After the SSL and a calling program have been loaded into main memory, SSL subroutines are mapped to the virtual address space of the task running that program. External references by the program to the mapped subroutines are then resolved by the operating system through a linkage table. The SSL facility reduces the space both on disk and in main memory occupied by copies of commonly used subroutines.

Stack
A line or list of elements in a pushdown storage device that handles data so that the next item to be retrieved is the one that has been most recently stored. The system stack on the VS is addressed by register 15.

Stack Header Block (SHB)
A data block describing the stack associated with one of a task's process levels. Entries in an SHB hold the stack vector and the address of the most recently-built JSCI save area. SHBs are referred to by the operating system to control stack switching.

Stack Header Block Table
A table of eight addresses, each pointing to the stack header block associated with one of a task's process levels.

Stack Limit Word
The address of the lowest byte location into which the stack may extend.

Stack Pointer
Contains the address of the current stack top. See also Stack Vector.

Stack Switching
The switching from one system stack area to another within a task's address space. There is a separate stack area associated with each of a task's process levels. A call to, or return from, an external subroutine can cause a change in a task's process level and stack area.

Stack Vector
: Holds a stack pointer and a stack limit word. On the VS, the system stack vector consists of general register 15 and control register 2. A program may use any two consecutive general registers (except the pair 15 and 0) as an additional stack vector.

Trap
: An unprogrammed conditional transfer of control to a specified address.

Translation RAM (T-RAM)
: A local page table, i.e., an area of local CP memory holding page frame numbers for the loaded portion of a task's virtual address space.

Virtual Memory or Virtual Storage
: An address space that does not correspond to the physical main memory addressing of the computer (and may be larger than the main memory available), a portion of which is mapped onto main memory in page size blocks (2K). This storage space may be used as addressable main memory by the user, as the computer handles all paging in and out of main memory automatically.

WCC
: Write Control Character for the VS workstation, the second byte of every WRITE command to the workstation. It controls locking, the alarm, the cursor, scrolling of the screen, and erasing.

Word
: On the VS, a sequence of 4 bytes, aligned on a 4-byte boundary.

INDEX

G

H

I

W

Z

WANG

Customer Comment Form

Publication Number 715-0422

Title VS PRINCIPLES OF OPERATION

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

- ☐ Support or Sales Rep
- ☐ Wang Supplies Division
- ☐ From another user
- ☐ Enclosed with equipment
- ☐ Don't know
- ☐ Other _____

How did you use this Publication?

- ☐ Introduction to the subject
- ☐ Classroom text (student)
- ☐ Classroom text (teacher)
- ☐ Self-study text
- ☐ Aid to advanced knowledge
- ☐ Guide to operating instructions
- ☐ As a reference manual
- ☐ Other _____

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Were there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

Were there any terms or concepts that were not defined properly? ☐ Y ☐ N If so, what were they? _____

After reading this document do you feel that you will be able to operate the equipment/software? ☐ Yes ☐ No ☐ Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) _____

Do you have any other comments or suggestions? _____

Name _____ **Street** _____

Title _____ **City** _____

Dept/Mail Stop _____ **State/Country** _____

Company _____ **Zip Code** _____ **Telephone** _____

Thank you for your help.

WANG

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WANG LABORATORIES, INC.
PUBLICATIONS DEVELOPMENT
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Fold

Cut along dotted line.

WANG

ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851
TEL. (617) 459-5000
TWX 710-343-6769, TELEX 94-7421

Printed in U.S.A.
715-0422
2-86