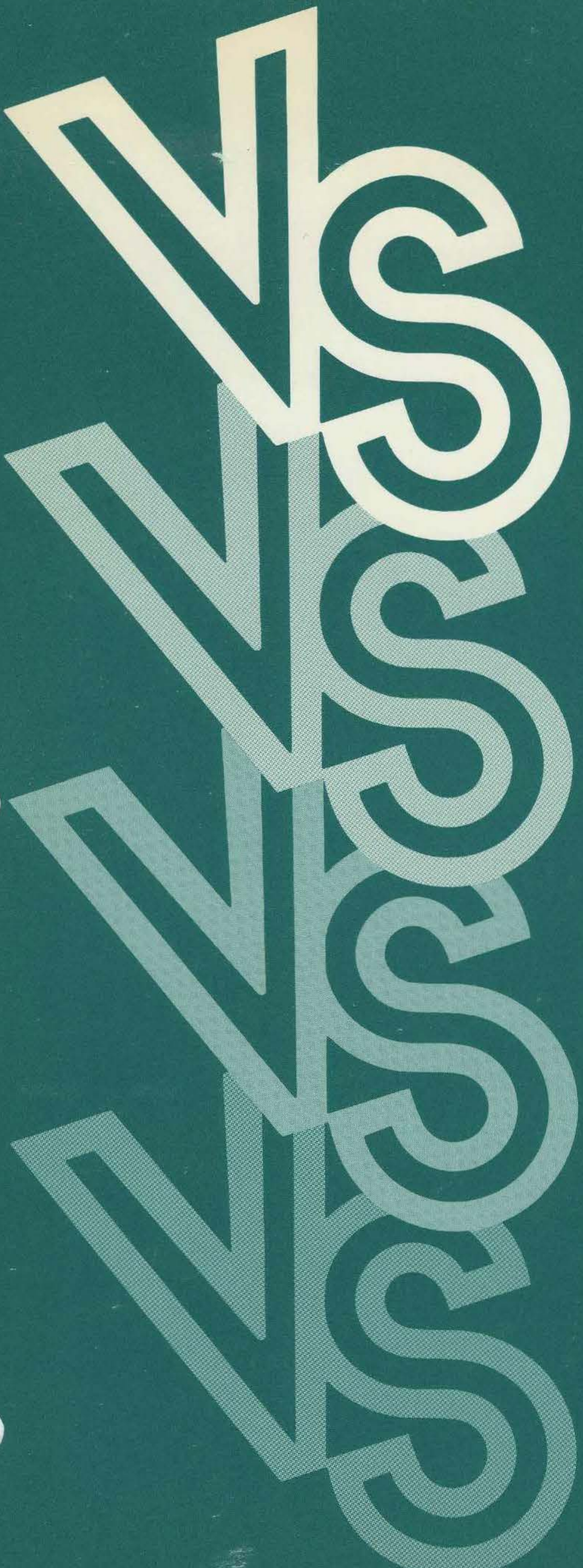


Principles of Operation

WANG

This manual is updated by: Addendum 800-1100PO-04.01

Principles of Operation

4th Edition — July, 1981

800-1100PO-04

WANG

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 this software package was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang Laboratories, Inc., or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the software package, the accompanying manual, or any related materials.

NOTICE:

All Wang Program Products are licensed to customers in accordance with the terms and conditions of the Wang Laboratories, Inc. Standard Program Products License; no ownership of Wang Software is transferred and any use beyond the terms of the aforesaid License, without the written authorization of Wang Laboratories, Inc., is prohibited.

This manual has been updated by Addendum 800-1100PO-04.01. For a list of these changes see the Summary of Changes.

WANG

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

PREFACE

This manual is intended as a reference tool for the VS programmer. Chapters 1 through 6 describe machine organization, general instruction and data formats, interrupts, and the functioning of the Central Processor (CP). Chapter 7 describes the format and operation of each instruction. Chapters 8 through 12 detail the interface with each type of Input/Output (I/O) device.

This document should be available at all VS sites for general reference. Additional information on the VS assembler language may be found in the *VS Assembler Language Reference Manual* (800-1200AS) and the *VS Assembler Language Pocket Guide* (800-6203AP).

This manual is updated by: Addendum 800-1100PO-04.01

SUMMARY OF CHANGES

1ST ADDENDUM TO THE 4TH EDITION OF VS PRINCIPLES OF OPERATION

Type	Description	Pages
New Features	Document History	DH-1
	New Tables and Figures:	
	. Diagrams of VS processors	1-2
	. Decimal floating-point number format	3-13
	. Physical address format	4-7
	. Virtual address format	4-8
	. Main memory page table entry format	4-9
	. SCR entry format	4-11
	. Address translation diagram	4-12
	. Operand M2 format for RRCB instruction	7-129
	. IODA for VS25 and VS100	8-2
	. VS25 and VS100 Fixed memory assignments	8-3
	. DA, IOP Status Tables for VS25 and VS100	8-4
	. I/O Command Table for VS25, VS100	8-4.1
	. Status Qualifier Byte for VS25, VS100	8-4.1
	. SIO, CIO, and HIO Instruction Format	8-4.3
	. R1 Format for SIO, CIO, and HIO	8-4.3
	. I/O Command Word (IOCW) Format	8-4.5
Documentation Additions	Manual now focuses on VS25, VS100 architecture:	
	Segment control registers (SCRs) added for address translation	2-1
	Translation RAM (T-RAM) structure of VS25, VS100	2-2
	New clock and comparator	2-3
	Decimal floating-point data representations	3-5
	Decimal floating-point instruction format	3-12.1
	Address translation for VS25, VS100	4-7
	Addressing exceptions for VS25, VS100	5-11
	VS100 machine check interruptions	5-14

Type	Description	Pages
Documentation Additions (cont'd.)	I/O Subsystem for VS25, VS100	8-1
	VS25, VS100 device addresses	8-2
	VS25, VS100 internal communications areas	8-3
	Status Qualifier Byte (SQB)	8-4.1
	I/O Command Table (IOCT)	8-4.1
	IOP, DA status tables (IOPST, DAST)	8-4
	SIO Instruction	8-4.4
	I/O Status Word (IOSW)	8-9
	Decimal floating-point instructions (new):	
	. AQ, AQR ADD DECIMAL (FLOATING-POINT)	7-4.1
	. CVP CONVERT DECIMAL (FLOATING-POINT) TO PACKED DECIMAL	7-40.1
	. CVQ CONVERT PACKED DECIMAL TO DECIMAL (FLOATING-POINT)	7-40.2
	. DQ, DQR DIVIDE DECIMAL (FLOATING-POINT)	7-52.1
	. LSCTL LOAD SEGMENT CONTROL REGISTER	7-96.1
	. MQ, MQR MULTIPLY DECIMAL (FLOATING-POINT)	7-110.1
	. SQ, SQR SUBTRACT DECIMAL (FLOATING-POINT)	7-162.1
	. STSCTL STORE SEGMENT CONTROL REGISTER	7-159.1
	Instructions modified:	
	. LPA LOAD PHYSICAL ADDRESS	7-93
	. RRCB RESET REFERENCE AND CHANGE BITS	7-129

This manual is updated by: Addendum 800-1100PO-04.01

CONTENTS

CHAPTER 1 INTRODUCTION TO THE WANG VS

CHAPTER 2 MACHINE ORGANIZATION

CHAPTER 3 DATA ORGANIZATION

CONTENTS (continued)

CHAPTER 4 INSTRUCTION EXECUTION

CHAPTER 5 INTERRUPTIONS

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

This manual is updated by: Addendum 800-1100PO-04.01

CONTENTS (continued)

This manual is updated by: Addendum 800-1100PO-04.01

CONTENTS (continued)

CONTENTS (continued)

APPENDICES

FIGURES

This manual is updated by: Addendum 800-1100PO-04.01

TABLES

TABLES (continued)

INTRODUCTION TO VS SYSTEMS

1.0 VS FAMILY CHARACTERISTICS

The Wang VS computer family consists of medium-scale, general-purpose computers designed to provide sophisticated hardware at a low cost. 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 8M bytes (M = 1,048,576); the addressing scheme allows a maximum memory size of 16M bytes. To help make full use of available memory for any VS system there is virtual memory support in the form of address translation hardware and several privileged instructions.

Input/output processors (IOPs) 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.

1.1 VS25 Characteristics

Basic Configuration

The VS25 is an entry-level VS system. The basic configuration of the VS25 consists of the CP and included 1.2M-byte diskette drive, main memory, an included fixed-disk drive, and an operator's 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 VS25.

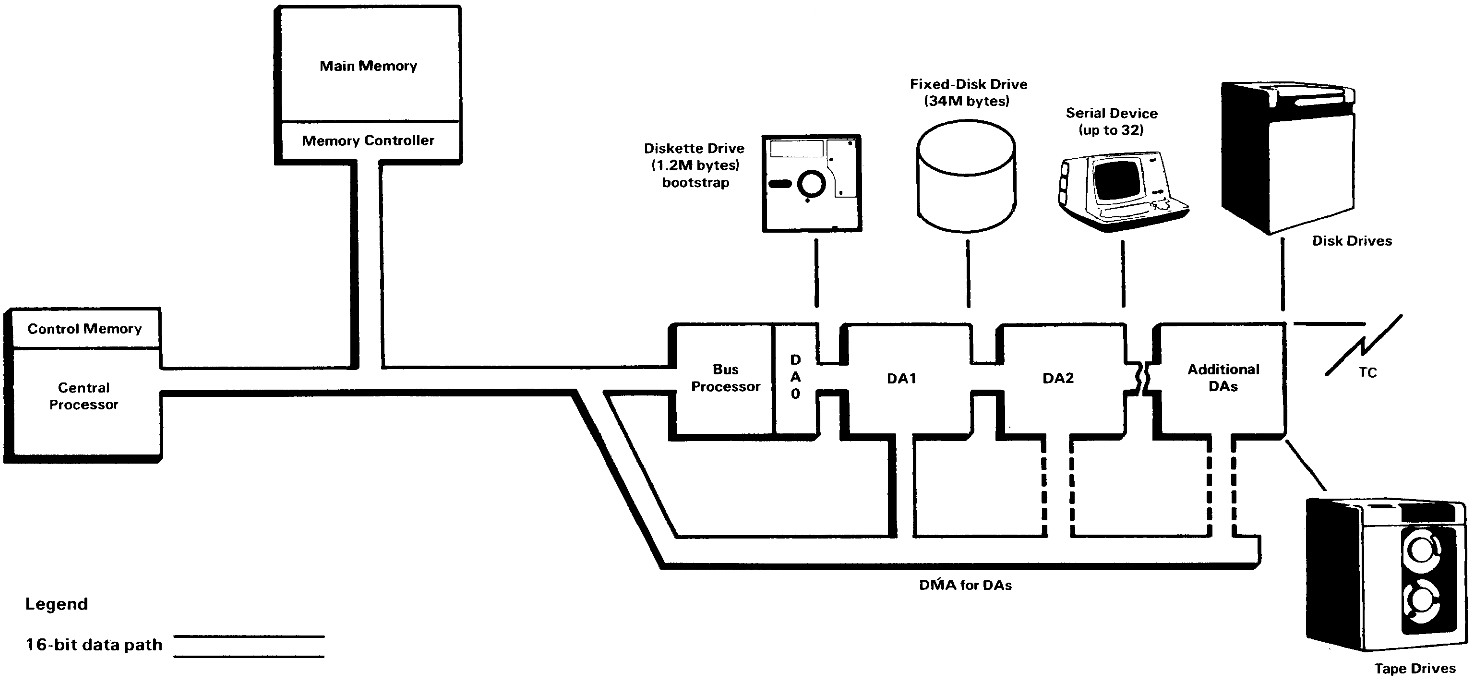


Figure 1-1. VS25 Architecture

1.2 VS80 Characteristics

Basic Configuration

The VS80 is the original VS system. The basic configuration of the VS80 consists of the CP and included 308K-byte diskette drive, main memory, one or more removable-disk drives, and an operator's console workstation. Additional I/O devices may be added as options.

Figure 1-2 is a diagram of the VS80.

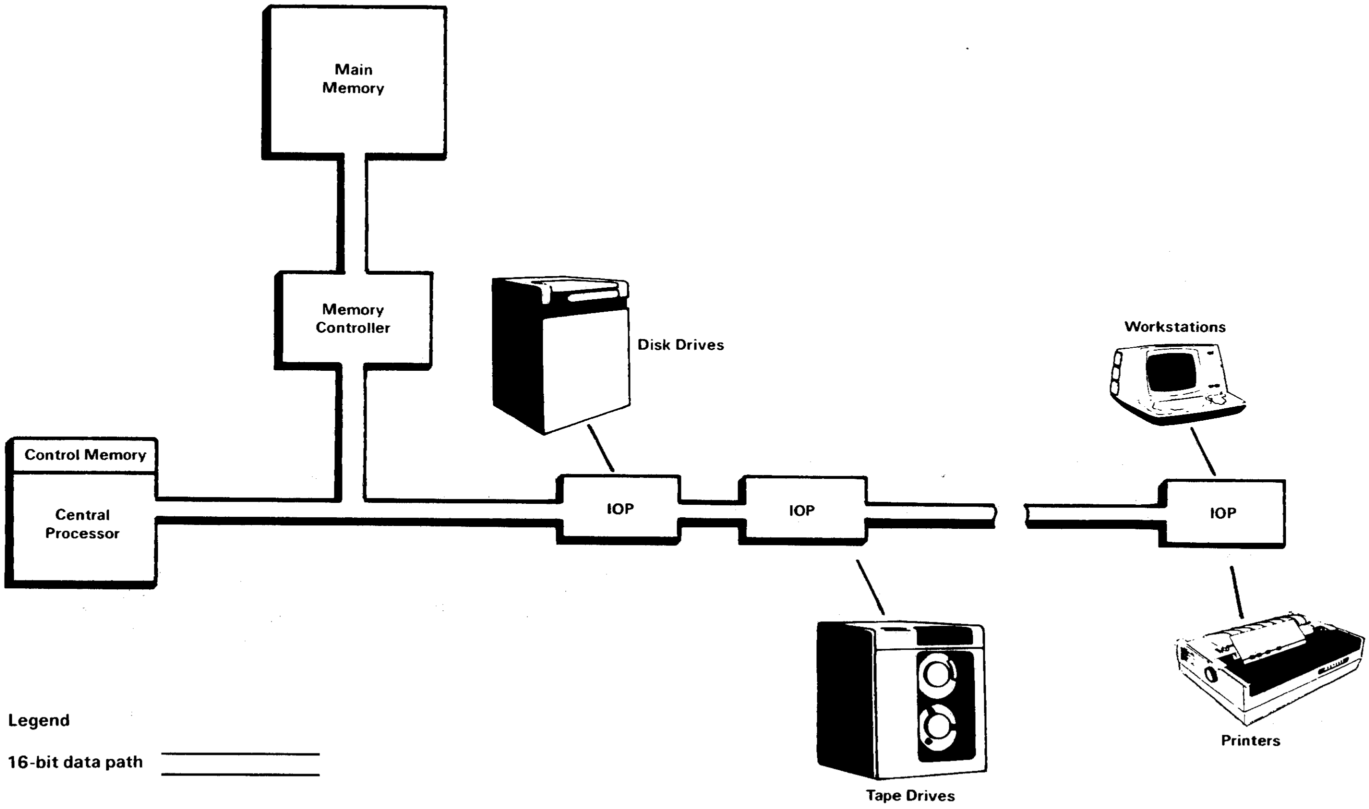


Figure 1-2. VS80 Architecture

1.3 VS100 Characteristics

Basic Configuration

The VS100 is the largest and fastest member of the VS family. The basic configuration of the VS100 consists of the CP, main memory, one or more removable-disk drives, and an operator's console workstation with attached 1.2M-byte diskette drive. Additional I/O devices may be added as options. Other VS systems having the same architecture, including the VS90, may exclude the cache memory feature.

Figure 1-3 is a diagram of the VS100.

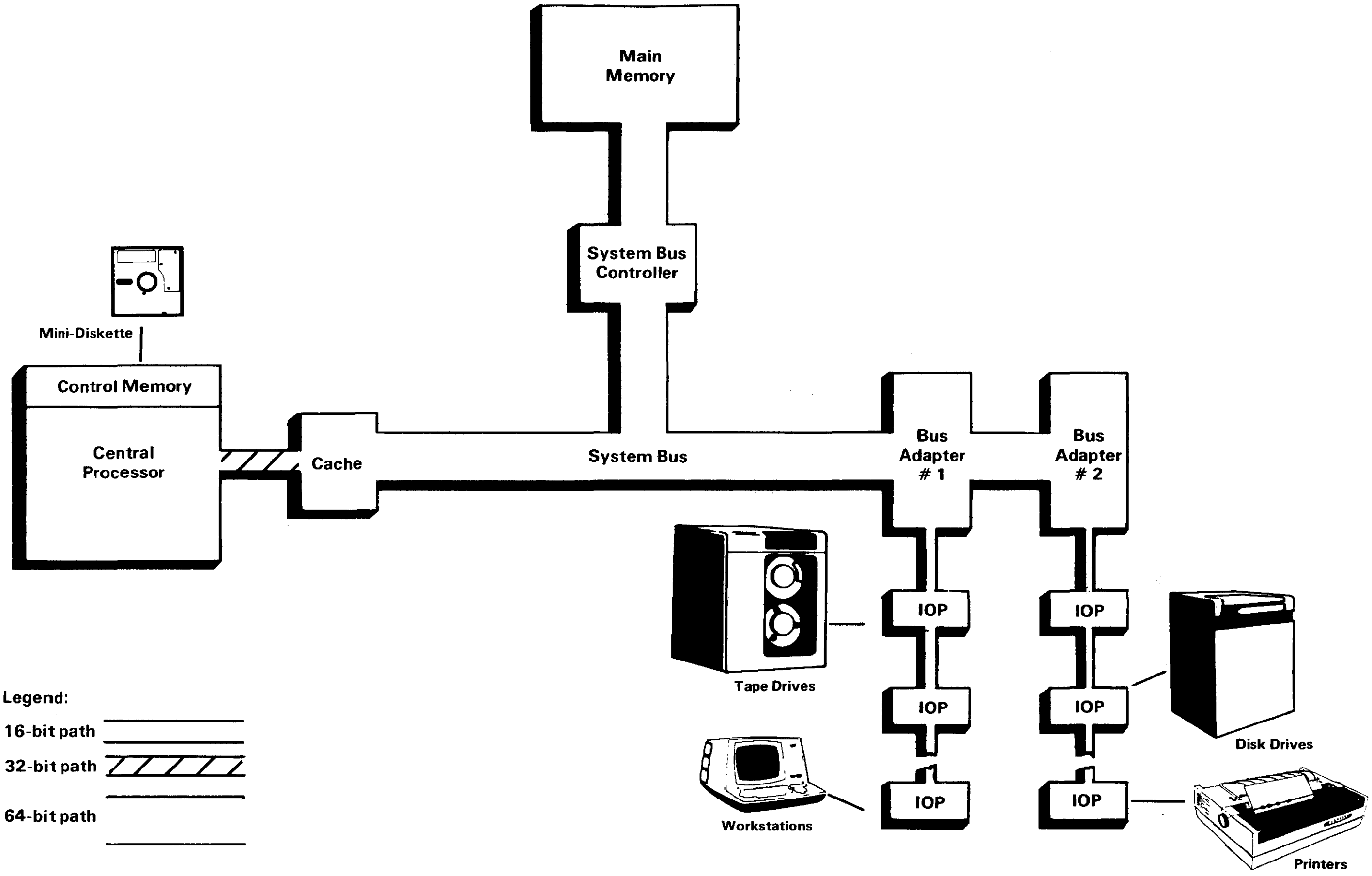


Figure 1-3. VS100 Architecture

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 provide for addressing multiple general registers by having several R fields.

2.1.2 Floating-Point Registers

Four floating-point registers, specified as registers 0, 2, 4, and 6, are provided. 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 (for the VS80: eight 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 the information is used. Control register allocations for the VS systems are as follows:

VS25, VS100	VS80	Allocation
CRO	CRO	High Range
CRI	CRI	Save Area Back Chain
CR2	CR2	System Stack Limit Word
CR3	CR3	Low Range
CR4	CR4	Modification Trap Address
CR5	CR5	Previous-Instruction Trap Address
CR6-11		Reserved
CR12-13	CR6	Time-of-Day Clock
CR14-15	CR7	Clock Comparator

Only the general structure of control registers is described here; a definition of the meaning of the register positions appears with the description of the facility 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 is associated with the stack handling facility and is referred to as the system stack limit word. Control registers 0 and 3-5 are associated with the debugging aids, and control registers 12-15 (for the VS80: control registers 6-7) are associated with the clock.

2.1.4 Translation RAM (T-RAM) -- VS25, VS100

A translation RAM (T-RAM) takes the place of LPTs for the VS25 and VS100, permitting the use of longer virtual and physical address spaces and enabling a more complex system of read and write protection. The T-RAM is a section of local CP memory (RAM) up to 8K bytes in size consisting of a halfword entry for each of the up to 4K pages of a potential 8M-byte virtual address space. A definition of the meaning of the entries appears with the description of address translation for the VS25 and VS100 in Subsection 4.3.3.

2.1.5 Local Page Tables (LPTs) -- VS80

There are three local page tables (LPTs) of one-byte entries, each associated with a valid memory segment. All memory references involve the translation of virtual memory addresses through use of one of these tables. The LPT for segment 0 is 128 entries long. The LPTs for segments 1 and 2 are each 256 entries long. A definition of the meaning of the entries appears with the description of address translation for the VS80 in Subsection 4.3.3.

2.1.6 Local Page Frame Table

There is one local page frame table, which contains two bits per page frame (2048 bytes on a 2048-byte boundary) of physical memory. Whenever some location in a page frame is referenced by a machine instruction, the reference bit of the corresponding local page frame table entry is set to 1. When this reference involves modification of the memory location, the change bit in the local page frame table entry is also set to 1. These entries are tested and reset by an Operating System Assist instruction.

2.1.7 Arithmetic and Logical Unit -- (ALU)

The arithmetic and logical unit (ALU) can process binary integers of fixed length, decimal integers of variable length, and logical information of either fixed or variable length. The ALU has a width of 16 bits for the VS25 and VS80, and a width of 32 bits for the VS100.

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. For the VS80, the cycle of the clock is approximately 994 days at 50 Hz and 828 days at 60 Hz. For the VS25 and VS100, which use a pair of registers for a counter, the cycle of the clock is meaninglessly large, in human terms.

The time-of-day clock for the VS25 and VS100 is a 64-bit binary counter (for the VS80: a 32-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 line frequency (1/50 or 1/60 second) for the VS80, split line frequency (1/100 or 1/120 second) for the VS25, or 2.5M Hz for the VS100.

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.

For the VS25 and VS100, the clock value resides in control registers 12 and 13 (for the VS80: in control register 6) and is set to zero during a power-on. 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 for the VS25 and VS100 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, a clock interruption is pending. The value of the clock comparator resides in control registers 14 and 15 and is set to all 1s during power-on. An interruption request disappears if the value in control registers 12 and 13 or control registers 14 and 15 is changed such that the value in control registers 12 and 13 is less than that in control registers 14 and 15. (For the VS80: control register 6 holds the clock value, and control register 7 holds the comparator value.) 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 VS25, a single bus processor (BP) controlling several device adapters (DAs) does the work of VS80 and VS100 IOPs. For the VS100, intelligent bus adapters (BAs) provide an interface between the CP and IOPs.

2.4 MAIN MEMORY

Main memory for all VS systems consists of semiconductor random access memory (RAM) with automatic with 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 4 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.

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.

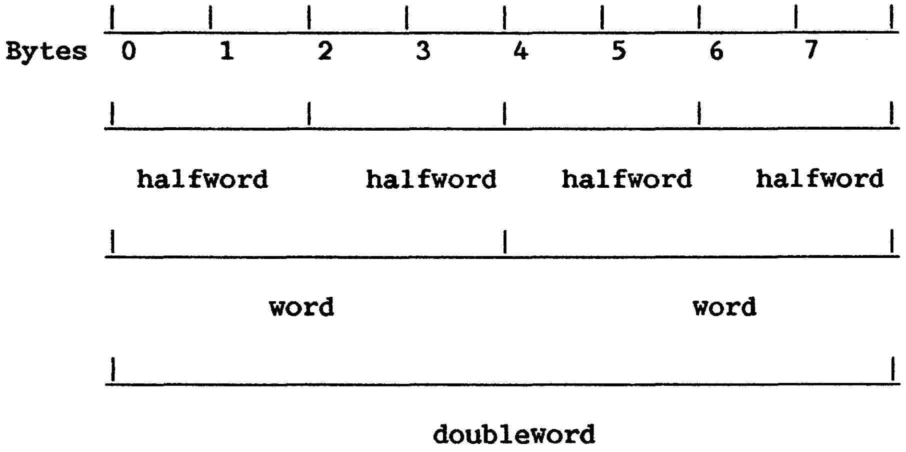


Figure 2-1. Sample Information Formats

2.4.2 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 binary 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 of this manual for details of addressing and address translation.

CHAPTER 3
DATA ORGANIZATION

3.1 INSTRUCTIONS--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)

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 Operation Code

In each format, the first instruction halfword consists of two parts. The first byte 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

The second byte is used either as two 4-bit fields or as a single 8-bit field. This byte can contain the following information:

4-bit operand register specification (R1, R2, or R3)

4-bit index register specification (X2)

4-bit mask (M1)

4-bit operand length specification (L1 or L2)

8-bit operand length specification (L)

8-bit byte of immediate data (I2)

4-bit stack vector specification (S).

In some instructions a 4-bit field or the whole second byte of the first halfword is ignored.

The second, third, and fourth halfwords may vary in format.

3.1.2 Operands

The VS allows up to three operands, depending on the instruction format. 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, specified by a bit mask, or 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 Subsection 4.2.1.

For purposes of describing the execution of instructions, operands are designated as first, second, and third operands. 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 participating in the addressing or execution part of an operation remain unchanged.

3.1.3 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-and-immediate-operand operation

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

RR Format--One Halfword

Op code	R_1	R_2
0 7	8 11	12 15

RL Format--Two Halfwords

Op code	R_1	L_2
0 7	8 11	12 31

RX Format--Two Halfwords

Op code	R_1	X_2	B_2	D_2
0 7	8 11	12 15	16 19	20 31

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

RS Format--Two Halfwords

Op code	R_1	R_3	B_2	D_2
0 7	8 11	12 15	16 19	20 31

RRL Format--Two Halfwords

```
 ____________________________________________________________
|               | R       M   | R      X  |         L        |
|   Op code     |  1 or   1   |  3 or  2  |         2        |
|_______________|_____________|___________|__________________|
 0            7 8          11 12        15 16               31
```

SI Format--Two Halfwords

```
 ____________________________________________________________
|               |      I      | B   |           D            |
|   Op code     |      2      | 1   |           1            |
|_______________|_____________|_____|________________________|
 0            7 8          15 16  19 20                     31
```

S Format--Two Halfwords

```
 ____________________________________________________________
|               |             | B   |         D              |
|     Op code   | ----------- | 1   |         1              |
|_______________|_____________|_____|________________________|
 0            7 8          15 16  19 20                     31
```

SS Format--Three Halfwords

```
 ____________________________________________________________
|               |             | B   | / /D | B     |/ / D   |
|    Op code    | L or L1/L2  | 1   | - - 1| 2     |- -   2 |
|_______________|_____________|_____|_/_/__|_______|/_/_____|
 0            7 8          15 16  19 20  31 32   35 36     47
```

SSI Format--Four Halfwords

```
 ________________________________________________________________
|          | L   |      | L     | / /B | / /D | / /B | / /D |
| Op code  | 1   |  I   | 2     | - - 1| - - 1| - - 2| - - 2|
|__________|_____|______|_______|_/_/__|_/_/__|_/_/__|_/_/__|
 0       7 8   15 16  23 24   31 32  35 36  47 48  51 52   63
```

Table 3-1 shows how the data formats for the following instructions are represented in VS memory. The relative addresses for a series of numbers are given in order to illustrate boundary alignments of fixed-point and floating-point data.

Table 3-1. 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	3C45600000000000	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

3.2 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.

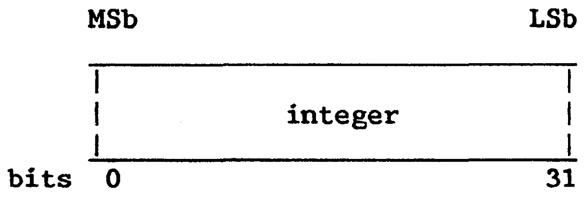


Figure 3-1. Fixed-Point Fullword Data Format

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 provides economy and 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. 2's-complement notation does not include a negative zero, so the set of negative numbers is 1 larger than the set of positive numbers.

3.3 DECIMAL INSTRUCTIONS

Decimal instructions provide 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 composing 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, indicating the number of additional bytes that the operand extends beyond the first byte.

The decimal arithmetic instructions provide for adding, subtracting, comparing, multiplying, and dividing, as well as for 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 3-2. The preferred sign codes are generated by all decimal arithmetic instructions.

Table 3-2. Bit Codes for Digits and Signs

Digit	Code	Preferred Sign Code	Allowed Sign Code
0	0000	- 1101	- 1011
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.

```
|-----MSB-----|_______ _ _ _ _____________________|-----LSB-----|
|      |      |      |       |      |      |      |      |      |
|digit |digit |digit |       |digit |digit |digit |digit | sign |
|______|______|______| _ _ _ |______|______|______|______|______|
```

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.

```
|-----MSB---|_____________ _ _ _ _____________|----LSB-----|
|     |     |      |      |      |     |      |     |      |
|zone |digit| zone |digit |      |zone |digit |sign |digit |
|_____|_____|______|______| _ _ _|_____|______|_____|______|
```

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 diagrammed 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.

```
|----MSB-----|_________________________________________|----LSB-----|
|            |            |            |            |            |
|   ASCII    |   ASCII    |            |   ASCII    |   ASCII    |
|   digit    |   digit    |  -------   |   digit    |   digit    |
|____________|____________|____________|____________|____________|
```

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 Subsection 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.

```
 ___________________________________________________   ______________
|   |                 |                            / /                |
| S | Characteristic  |      14-digit Fraction                        |
|___|_________________|___________________________/ /_________________|
 0                     8                                            63
```

```
 ______________________________________   ______________
|   |                 |               / /               |
| S | Characteristic  |  6-digit Fraction               |
|___|_________________|______________/ /________________|
 0                     8                              31
```

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 MULTIPLY, 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 (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, or (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 non-0 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 Subsection 3.1.3. 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 participating 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 as follows:

```
          _____________________________________________  ______
          | |                 |                       / /      |
          |S| Characteristic  |     14-digit decimal Fraction  |
          |_|_________________|_______________________/ /______|
     bits  0 1                8                                63
```

Figure 3-5.1. 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 arthmetic may use the four 8-byte floating-point registers for data manipulations. Decimal floating-point instructions provide both normalized RR and normalized RX formats for arithmetic operations--i.e., 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 3-6 shows the structure of fixed-length logical operands.

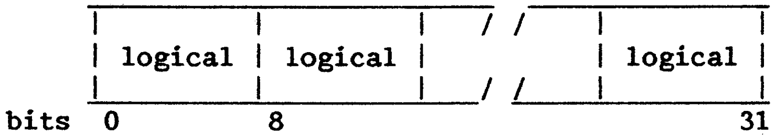


Figure 3-6. Fixed-Length Logical Operand (one to four bytes)

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 3-7 shows the structure of variable-length logical operands.

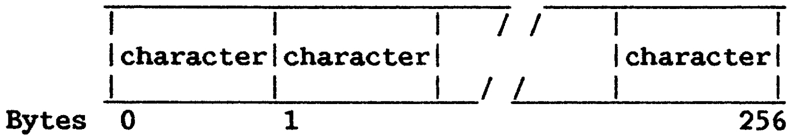


Figure 3-7. Variable-Length Logical Operand (up to 256 bytes)

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 LINKED LIST INSTRUCTIONS

The instructions ENQ, ENSK, DEQ, and DESK are provided to handle lists of blocks connected by pointers. Two kinds of linked lists are supported: first-in first-out (FIFO) lists, and last-in first-out (LIFO) lists. The instructions provide the means to add to and delete from the lists, and to determine whether the lists are empty or not.

3.6.1 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 3-8 is a diagram of such a list. The pointers in the blocks are all at a displacement into the block determined by the ENSK or DESK instruction's displacement field.

```
                First Block       Second Block                Last Block
       ______   ___________       ___________                 ___________
      |Head|----|         |    ---|         |   |       | ---|         |
      |Ptr_|    |         |   /   |         |   |       |/   |         |
                |_________|  /    |_________|  /|       |    |_________|
                |Next Ptr|---     |Next Ptr|--- |       |    |____0____|
                |         |       |         |   | . . . |    |         |
                |_________|       |_________|   |       |    |_________|
```

Figure 3-8. LIFO List

3.6.2 Structure of FIFO Lists

The FIFO list, pictured in Figure 3-9, 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.

```
                First Block       Second Block                Last Block
                                                |       |
       ______   ___________       ___________   |       |     ___________
      |Head|----|         |    ---|         |   |       | ---|         |
      |Ptr_|    |         |   /   |         |   |       |/ / |         |
      |Tail|    |_________|  /    |_________|  /|       | /  |_________|
      |Ptr_|--  |Next Ptr|---     |Next Ptr|--- |       | |  |____0____|
            |   |         |       |         |   | . . . | |  |         |
            |   |_________|       |_________|   |       | |  |_________|
            |                                             |
            |_____________________________________________|
```

Figure 3-9. FIFO List

3.7 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 3-10. 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 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.

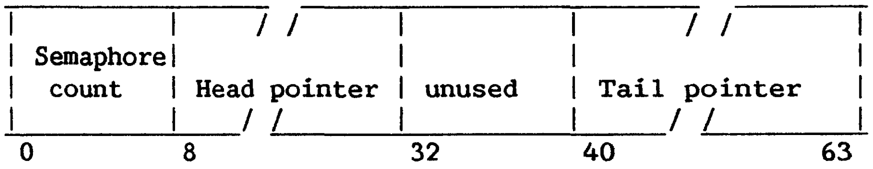


Figure 3-10. Semaphore

3.8 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, i.e., 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 (i.e., 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 0 whenever this word is modified by one of these instructions. The stack limit word is unchanged by these instructions.

CHAPTER 4
INSTRUCTION EXECUTION

4.1 PROGRAM CONTROL WORD

The Program Control Word (PCW), eight bytes long, 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 detailed 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 or a HALT I/O command is received.

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, and a 1-byte program mask field, with one byte reserved for later options. The PCW for a program can be inspected through Debug mode or by doing a program dump. Figure 4-1 shows the PCW format.

```
 ____________________________________________________________________
|                 |
| Interruption    |          Current  instruction  address
|     code        |
 --------------------------------------------------------------------
0                 8                                                 31
```

```
 ______________________________________________________________
      |   |   |///|///|   |   |   |   |   |   |   |   |   |////
   W  |C  |P  |///|///|I  |T  |M  |B  |D  |E  |S  |EM |BT |////
      |   |   |///|///|   |   |   |   |   |   |   |   |   |////
 --------------------------------------------------------------
32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47
```

Status Field

```
 ____________________________________________________________
         |F |   |   |   |/////|///////////////////////////|
   CC    |P |DO |EU |SG |/////|///////////////////////////|
         |O |   |   |   |/////|///////////////////////////|
 ------------------------------------------------------------
48  49  50  51  52  53  54  55  56                         63
```

Program Mask Field

Figure 4-1. PCW Format

Following is a more detailed explanation of the function of each bit in the PCW.

PCW Bits	Mnemonic	Function
0-7		Interruption code
8-31		Current instruction address
Status Field (system mask)		
32	W	Wait state 0 = Operating state 1 = Wait state
33	C	Control mode 0 = Normal operating mode 1 = Control mode
34	P	Memory protection violation and privileged instruction trap 0 = Do not trap on memory protection violation or privileged instruction 1 = Trap on memory protection violation or privileged instruction

PCW Bits	Mnemonic	Function
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
Status Field		
40	B	PCW single address compare trap 0 = No PCW single address compare trap in effect 1 = Trap on PCW single address compare equal
41	D	Single byte modification trap 0 = No single byte modification trap in effect 1 = Trap on unequal compare with byte at specified byte
42	E	PCW range trap 0 = No PCW range trap in effect 1 = Trap on unequal compare with byte at specified PCW range
43	S	Single step trap 0 = No step exception 1 = Trap after execution of next instruction
44	EM	Extended modification trap
45	BT	Branch-Taken trap
46-47		Reserved
48-49	CC	Condition code

PCW Bits	Mnemonic	Function
Program Mask Field		
50	FPO	Fixed-Point overflow mask 0 = Do not interrupt on overflow 1 = Overflow will cause interruption
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 instructions) 0= Do not interrupt on overflow 1= Overflow will cause interruption
54-55		Reserved
Reserved Byte		
56-63		Reserved

4.1.1 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 7.

4.2 ADDRESSING

4.2.1 Base-Displacement Address Generation

Base-displacement, relative, and direct address generation are all available, as described below.

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 to provide for the flexible specification of input, output, and working areas, all instructions referring to main memory can employ a full address.

The address used to refer to main memory is 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 as a means of 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, i.e., 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 be used to specify one of many items associated with an element. In processing records, the displacement can be used to 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 necessarily 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.

As an aid in describing the logic of the instruction format, examples of two instructions and their related instruction formats follow.

RR Format: LR 7,9

```
 __________________________________
|               |       |       |
|   LOAD (18)   |   7   |   9   |
|_______________|_______|_______|
0               8      12      15
```

Execution of the LOAD instruction copies the contents of general register 9 to general register 7.

RX Format: ST 3,TOTAL

```
 ___________________________________________________________________
|               |       |       |       |                           |
|   STORE (50)  |   3   |  10   |  14   |            300            |
|_______________|_______|_______|_______|___________________________|
0               8      12      16      20                          31
```

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.

4.2.2 Relative Address Generation

For relative addressing 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 if, for example, 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.

Otherwise, the rules for relative address generation are the same as the rules for base-displacement address generation.

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 IOCA locations.

4.3 ADDRESS TRANSLATION

Address translation is the process of converting virtual addresses, referring to a user's virtual address space, into physical addresses, referring to main memory locations. This conversion is accomplished without the user's knowledge, by a combination of hardware and operating system action.

4.3.1 Physical/Virtual Address Space

Main memory for all VS machines consists of byte-addressable random access memory (RAM). It is spanned by a 24-bit address, allowing for up to 16M bytes of addressable storage (i.e., 2**24 = 16M). Main memory addresses consist of a 13-bit page frame number and an 11-bit byte index to locations within the page, as illustrated in Figure 4-2, below. The range of main memory addresses depends upon the amount of physical memory configured into the installation, and currently varies from 256K bytes to 8M bytes with different processors of the VS family. (Some VS80 configurations use 128K bytes of memory.) For the VS25 and VS80, byte-aligned write operations of 1 or 2 bytes and halfword-aligned read operations of 2 bytes are supported. For the VS100, byte-aligned write operations of 1, 2, 4, or 8 bytes are supported, along with doubleword-aligned read operations of 8 bytes only.

Main memory, located on semiconductor chips in the CP cabinet, is divided logically into page frames of 2K bytes, each aligned on a 2K-byte boundary and containing exactly one page of information.

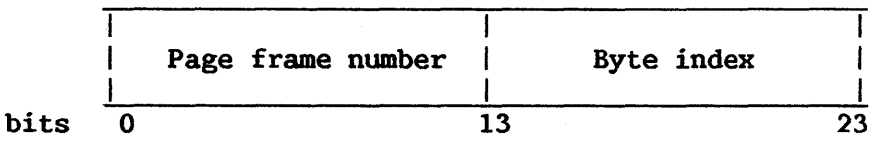


Figure 4-2. Physical Address Format

Virtual memory, located in disk storage, is divided logically into pages and segments. 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, below. Virtual pages also are 2K bytes in size, beginning on a 2K-byte boundary; physically, each page occupies one sector of a disk platter. Segments are blocks of pages beginning on a 1M-byte boundary. Segment 0 is 256K bytes in size for the VS80, and 1M bytes for VS25 and VS100 systems; segments 1 and 2 are each up to 512K bytes for the VS80, and up to 1M bytes for the VS25 and VS100. Pages of virtual memory are copied as needed into available page frames of main memory, as discussed in Subsection 4.3.2.

```
      _________________________________________
     |                      |                  |
     | Virtual page number  |    Byte index    |
     |______________________|__________________|
bits  0                     13                 23
```

Figure 4-3. Virtual Address Format

Bits 0-3 of Figure 4-3 are the segment index of the virtual address.

4.3.2 Overview of 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, the initials "VS" stand 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 2K bytes, or one page, by a dedicated operating system task 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 although 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.

This manual is updated by: Addendum 800-1100PO-04.01

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.

4.3.3 Details of Address Translation

Main Memory Page Tables

An essential part of the address translation mechanism is a task's page table, a section of main memory that defines the mapping of each task's virtual address space to main memory page frames. The format of page table entries is illustrated in Figure 4-4, below.

```
       _____________________________
       | | | |                     |
       |F|R|W|  Page frame number  |
       | |P|P|_____________________|
bits    0 1 2 3                   15
```

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 3-15 are its page frame number. If not, F=1 and the rest of the entry is not examined.

Bits 1 and 2 (RP and WP) indicate whether read and write protection, respectively, are in effect for the page. Together, these two bits define four protection classes and two protection states, as follows:

RP	WP	Meaning
0	0	Unprotected
0	1	No write allowed in user state
1	0	No read, write, or execute allowed in user state
1	1	No write allowed in system state or user state

Although this scheme allows protection to vary from one page to another, pages are currently given the same protection for an entire segment.

The first step of address translation is to find out whether the main memory page table contains a page frame number for the virtual address. If so, the CP concatenates this 13-bit number with the 11-bit virtual byte index (offset) to form a 24-bit physical address, as illustrated in Figure 4-6, below; it then uses this physical address to access the data in memory. If not, 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 task's page table. The task can then be dispatched again.

Local Page Table (T-RAM)

To speed up address translation, a subset of the currently executing task's main memory page table is held in local CP memory and is checked first during translations. Most instances of address translation are accomplished by this translation RAM (T-RAM), a CP local page table of 4K entries representing 4K*2K or 8M bytes of virtual address space.

At the start of each user's time slice, the fault bit of each T-RAM entry is set to 1, indicating that they do not hold page frame numbers. Then, as pages are referenced by the task, page frame numbers found in the main memory page table 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 (times estimated for the VS25) required for a main memory page table access. (For the VS80, a local page table for each segment of a user's address space contains a one-byte page frame number for each page currently residing in main memory. VS80 local page tables are loaded at the start of each user's time slice by privileged assembler instructions LPT0, LPT1, and LPT2.)

Segment Control Registers (SCRs)

The address of a task's main memory page table for a segment is loaded into the privileged segment control register (SCR) for the segment at the beginning of the task's time slice. SCR format is illustrated in Figure 4-5, below. Address translation requires a prior look-up of the page table address held in the SCR indicated by the segment portion of the faulting virtual address, i.e., by bits 0-3 of Figure 4-3, above.

NOTE

SCRs 0-7 each control a segment having a maximum size of 1M bytes of virtual address space (i.e., having 512 entries in its page table). Therefore, the 8 SCRs support a maximum contiguous address space of 8M bytes.

The appropriate SCRs for each task contain several pieces of information, arranged in the following format:

	M	Length-1	Page table address/8	V P
bits	0	1	10 30	31

Figure 4-5. Segment Control Register (SCR) Entry Format

Bit 0, the M bit, indicates whether monitoring is in effect for the segment. When it is in effect (i.e., when M=1), each page table entry loaded into the T-RAM from the main memory page table pointed to by the SCR is also listed in the monitor area to facilitate the eventual clearing of the T-RAM. Refer to Subsection 4.3.4, below, for a discussion of the monitor area.

Bits 1-9 represent the length, minus 1, of the page table. Therefore, a full page table of 512 halfword entries, corresponding to a fully utilized segment or region of 512 2K-byte pages, would have bits 1-9 all set to 1 in its SCR.

Bits 10-30 of Figure 4-5 represent the page table address divided by 8, i.e., lacking three low-order 0s (because each page table begins on at least a doubleword boundary). Restoring these 0s gives a full 24-bit address, which may be either a virtual or a physical address accordingly as bit 31, the VP bit, is 0 or 1. If it is 1, the address of the appropriate page table entry is just the page table address reported in bits 10-30, plus twice the virtual page index (reported in bits 4-12 of Figure 4-3, above). That is, the page table contains a halfword entry for each page represented.

If the VP bit of the SCR is 0, then the page table address reported in bits 10-30 of the SCR is itself a virtual address and must be translated using a second SCR. This second SCR must contain a physical address for the page table. (If it does not, an SCR recursion exception is noted; refer to Subsection 5.8.2, below, for details.) Whether the physical address of the page table is obtained using one or two SCRs, the result is the same: the address of the appropriate page table entry is calculated, and the entry is forwarded to the CP so it can continue executing the user program. The entry is also written into the T-RAM, so that subsequent references to the page can be satisfied more quickly.

SCRs are loaded and stored using the privileged LSCTL and STSCTL instructions. Refer also to the descriptions of these instructions in Chapter 7 of this manual.

Figure 4-6, below, is an illustration of the address translation process.

```
                  Virtual Address
______________________________________________________
|              |               |                     |
|   Segment    |  Page index   |      Byte index     |
|    number    |            .  |                     |
______________________________________________________
        |                                 |
        |                                 |
        |                                 |
        V     SCR                         |
________________________________          |
| Page table   |               |          |
|  physical    |  Page table   |          |
|   address    |     length    |          |
________________________________          |
        |                                 |
        | + (page index)*2                |
        |                                 |
        V Page Table Entry                |
________________________________          |
|   |                          |          |
|   |    Page frame number     |          |
|   |                          |          |
________________________________          |
        |                                 |
        |                                 |
        |                                 |
        V         Physical Address        V
______________________________________________________
|                              |                     |
|      Page frame number       |     Byte index      |
|                              |                     |
______________________________________________________
```

Figure 4-6. Virtual-to-Physical Address Translation

4.3.4 T-RAM Monitor Area

The monitor area makes possible the efficient clearing of T-RAM entries. The monitor is an area of local CP memory recording the virtual page numbers of pages loaded into main memory during the current time slice; only the corresponding T-RAM entries need be cleared for the start of a new time slice. For further discussion of the T-RAM monitor area and its use during address translation, refer to the description of the RRCB instruction in Chapter 7 of this manual.

4.3.5 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 8K entries of 2 bits each. These are a reference bit and a change bit for each of the up to 8K addressable pages 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 system paging task uses the reference and change bits along with an aging count in deciding which virtual pages to overwrite with new ones during paging operations. For details of RCT use, refer to the description of the RRCB instruction in Chapter 7 of this manual.

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 0, 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. Note that the relation of the two formats in branch-address specification is not the same as in operand-address specification. For operands, the address specified by X1, B2, and D2 is the operand address, but the register specified by R2 contains the operand itself.

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. Five classes of interruption conditions are possible: I/O, clock, program, supervisor call, and machine check.

Each class of interruption except supervisor call 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. The supervisor call class of interruption has only a new PCW in a permanently assigned main memory location. The supervisor call old PCW is stored on the top of the system stack, as addressed by general register 15. (See stack-oriented instruction descriptions in Chapter 7.)

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 higher priority interruption has been serviced.

5.2.1 Instruction Execution

An interruption occurs between instructions, except for interruptible instructions, as explained in the preceding paragraph. 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, 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.

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

5.2.2 Classes of Interruptions

The five 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 of the old PCW and in some cases by additional information placed in main memory during the interruption. The bits of the interruption code are numbered 0-7, according to their position in the PCW.

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 8.6.) For program interruptions, additional information may be provided in the form of a segment index and a page index stored in the page fault reporting area on an address translation exception. 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 (decimal)	Length (decimal)	Function
0	8	Input/Output Status Word (IOSW)
8	24	reserved for control mode
32	8	Old PCW for machine check
40	8	New PCW for machine check
48	8	Old PCW for program check
56	8	New PCW for program check
64	8	Old PCW for clock interrupt
72	8	New PCW for clock interrupt
80	8	Old PCW for I/O interrupt
88	8	New PCW for I/O interrupt
96	8	New PCW for SVC
104	10	Unused--available for software use
114	2	Page fault reporting area
116	12	Machine check reporting area
128	Variable	I/O Command Address area (IOCAs)

NOTE

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

5.2.3 Location Determination

In general, the instruction causing the interruption is given by the address in the PCW. 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 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.

5.4 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 turning 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 32-bit binary quantities.

The loading of a clock comparator value that is already less than or equal to the time-of-day clock value causes an immediate interruption; loading control register 7 with a value greater than that in control register 6 resets any pending clock interrupt.

5.5 PROGRAM INTERRUPTION

Exceptions resulting 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.

If the new PCW for a program interruption has an unacceptable instruction address, another program interruption occurs. Since this second program interruption introduces the same unacceptable instruction address, a string of program interruptions is established that may be broken only by an I/O interruption. If these interruptions also have an unacceptable new PCW, new supervisor information must be introduced by initial program loading or by manual intervention.

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 Subsection 8.6.2).

5.5.1 Program Interruption Codes in the PCW

Programming Errors and Miscellaneous Exceptions	Hex Code
General	
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
Debugging Aids	
PCW trap	10
Virtual destination trap	11
Branch-Taken trap	12
Single-Step trap	13
Address Translation Exception	20
Paging File I/O Error (software-defined error code)	28
Unresolved External Reference (software-defined error code)	29
Stack Facility	
Stack overflow	30
Floating-Point Exceptions	
Floating-Point overflow	40
Floating-Point underflow	41
Significance	42
Floating-Point divide	43

5.5.2 Access Exceptions

The protection, addressing, PCW trap, virtual destination trap, segment fault, page translation, and page fault 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. Additionally, each instruction can cause an access exception to be recognized due to instruction fetch.

Programming Note: An access exception is indicated only if the instruction with which the exception is associated is executed. In particular, 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 or an interruption changes the instruction sequence such that the inaccessible data is not required.

5.6 PROGRAMMING ERRORS AND MISCELLANEOUS EXCEPTIONS

5.6.1 Operation Exception

When an operation code is not assigned, an operation exception is recognized. For the purpose of recognizing an operation exception, the first eight bits of an instruction form the operation code.

5.6.2 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, LPTO, LPT1, LPT2, LPCW, RRCB, STNSM, STOSM, SIO, STDD, and SVCX. When a privileged instruction is encountered while this bit (also known as the memory protection violation bit, or the privileged-instruction trap bit) is on in the PCW, a privileged-operation exception is recognized, and the instruction is suppressed.

5.6.3 Execute Exception

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

5.6.4 Protection Exception

When the address of a receiver operand in memory is in a protected segment of memory (segment 0 or 1), a protection exception is recognized if the memory protection violation and privileged instruction trap bit is on in the PCW.

5.6.5 Addressing Exception

When an address specifies any part of a datum, an instruction, or a control word outside the available memory for the particular installation, 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.

5.6.6 Specification Exception

A specification exception is recognized when any of the following holds true:

1. An operand address does not designate a location on a doubleword, word, or halfword boundary, depending on the instruction type.

2. The first operand field is shorter than or equal to the second operand field in decimal division.

3. An invalid head/tail queue word has been specified in enqueue/dequeue operations.

4. Other special cases exist.

5.6.7 Data Exception

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

1. The digit codes of operands in decimal arithmetic or editing operations or in CONVERT TO BINARY are incorrect

2. Fields in decimal arithmetic overlap incorrectly.

5.6.8 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.

5.6.9 Fixed-Point Divide Exception

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

1. The quotient exceeds the register size in fixed-point division, including division by 0

2. The result of CONVERT TO BINARY exceeds 31 bits.

5.6.10 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.

5.6.11 Decimal Divide Exception

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

5.6.12 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.

5.6.13 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.

5.7 DEBUGGING AIDS

5.7.1 Modification Trap Feature

The modification trap, when enabled, causes a target, i.e., a general or floating-point register or a field of up to 64 bytes in memory, to be compared with a comparand at the beginning of every machine instruction; if they are not equal, a program interruption (code X'11') occurs and the instruction is suppressed. For interruptible instructions, the comparison occurs before each execution unit of the instruction.

There are two forms of this trap, both of which use the modification trap control register (control register 4). These forms, therefore, should not be enabled at the same time.

Single-Byte Modification Trap

This trap is enabled by setting PCW bit 41. The target is always a 1-byte field in memory. The target and comparand are specified by loading the trap control register with the comparand (high-order byte) and the address of the target (low-order three bytes).

Extended Modification Trap

This trap is enabled by setting PCW bit 44. The target and comparand are specified by loading the trap control register with the virtual address of a Trap Descriptor Area, which must be fullword aligned.

The Trap Descriptor Area has the following format:

Byte 0 Flag and length

Bits 0-1 Target select:

00	Memory
01	(Invalid)
10	General register
11	Floating-Point register

Bits 2-7 Comparand and length minus 1
(ignored if the target is a register)

Bytes 1-3 If the target is a register:
register number (low bits of byte 3)

If the target is in memory:
virtual address of the target

Bytes 4-nn Comparand value
(The comparand is the same
length as the target)

The trap control register can be altered and the trap enabled or disabled only by privileged instructions.

If the target is a memory operand, an access (fetch) interruption will be taken on any part of the operand that is invalid when the trap is checked. For the extended modification trap, an access interruption will also be taken on any part of the Trap Descriptor Area that is invalid. If the address of the Trap Descriptor Area is not fullword aligned, or an invalid target type is specified, or (for the floating-point register trap) an invalid floating-point register number is specified, a specification interrupt is taken.

If both modification traps are enabled, the single-byte trap will be checked before the extended trap.

5.7.2 PCW Trap Feature

When the PCW trap is enabled, before each instruction is executed a check is made to determine if the address of the instruction's first byte is in the specified range. If it is, the instruction is suppressed, the PCW address is not updated, and a program interruption (code X'10') is taken.

There are two forms of this trap; since they both use the low-range control register (control register 3) they should not both be enabled at the same time.

Single-Address PCW Trap

This trap is enabled by setting PCW bit 40. The range for which to check is always a single address contained in the low-range control register.

PCW Range Trap

This trap is enabled by setting PCW bit 42. If the high-order byte of the high-range control register (control register 0) is 0, the range is from the address in the low-range control register to the address in the high-range control register. If the high-order byte of the high-range control register is nonzero, the range is from address 0 to 1 less than the address in the low-range control register, and from 1 more than the address in the high-range control register to the highest possible address. The address in the low-range control register should not be greater than the address in the high-range control register.

The trap control registers can be altered, and the trap can be enabled or disabled, only by privileged instructions.

If both PCW traps are enabled, the single-address trap is checked before the range trap.

The PCW trap is checked after the modification trap.

5.7.3 Branch-Taken Trap Feature

When the branch-taken trap feature is enabled (by setting PCW bit 45), a program interruption (code X'12') will occur after the execution of any instruction that has successfully loaded the PCW address field. The current PCW address is the branch address set by the instruction; the previous-instruction-address control register (control register 5) will contain the address of the branch instruction. The branch-taken trap is not taken on instructions which modify the entire PCW (LPCW, SVC, or SVCX).

This trap may be enabled or disabled only by privileged instructions.

The branch-taken trap is checked before the single-step trap and the timer and I/O interrupt condition.

5.7.4 Single-Step Feature

A single-step trap can be enabled by turning on the trap bit in the PCW (PCW bit 43). This guarantees an interruption (code X'13') after execution of the next instruction has been completed. (For an interruptible instruction, this interrupt will occur only following the final execution unit of the instruction.) If any other program interrupt condition occurs during the execution of the instruction, it will take precedence over the single-step trap condition (since this trap can be inferred from presentation of any program interrupt). If this trap is taken, the PCW address field will contain the address of the next instruction.

For instructions that modify the system status or debug status bytes of the PCW (LPCW, STOSM, STNSM, SVC, or SVCX), the trap will be taken if the trap bit is set in the PCW before the instruction is executed. It will also be taken by SVCX if the trap bit is set in the new PCW loaded by that instruction.

The single-step trap is checked after the branch-taken trap, and before the timer and I/O interrupt conditions. It also takes precedence over a wait state PCW, which would have been introduced by the stepped instruction.

5.7.5 Previous Instruction Address Feature

If any of the PCW debug trap bits (bits 40-45) are set at the start of an instruction, the current instruction address is stored in the previous-instruction-address control register (control register 5) after all of the debugging traps are checked. If any of the debugging traps are taken after execution of this instruction and before execution of the next instruction, the previous-instruction-address control register will hold the address of the last instruction executed; the PCW address is the address of the instruction about to be executed.

Programming Notes

After a branch-taken trap, control register 5 holds the address of the branch instruction.

After a modification trap, control register 5 holds the address of the instruction that modified memory or registers. If that was an interruptible instruction and was not the last unit of execution, the PCW address and the address in control register 5 both point to the start of the interruptible instruction.

5.8 ADDRESSING EXCEPTIONS

5.8.1 Address Translation Exceptions

Three address translation exceptions, all having an interruption code of X'20', can occur in the course of address translation. All three cause the segment and page index of the virtual address to be written to the page fault reporting area, i.e., to location X'72' of main memory. A segment fault exception is the first, and occurs when the segment index of a virtual address is not valid. Currently, only segment indexes of 0, 1, and 2 are valid.

A page translation exception is the second, and occurs when twice the page index of a virtual address is greater than or equal to the page table length indicated in the SCR for the segment. In this case there is no page table entry corresponding to the virtual page.

A page fault exception is the third, and 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.

5.8.2 Page Table Address Exceptions

Two additional classes of exception exist in association with the address translation mechanism. The page table address fault exception, with an interruption code of X'21', is the first of these, and occurs when the page table address reported in the appropriate SCR is virtual and is faulted.

The SCR recursion exception, with an interruption code of X'22', is the second, and occurs when the page table address reported in the appropriate SCR is virtual and points to a second SCR, but the second SCR also contains a virtual address rather than a physical address. In this case the second virtual address is not translated and an exception is noted immediately.

5.9 STACK OVERFLOW EXCEPTION

This exception is unique to the stack-oriented instructions and may occur during any of these instructions. A stack overflow interruption occurs under either of the following conditions:

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

2. 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.

5.10 FLOATING-POINT EXCEPTIONS

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

5.10.1 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.

5.10.2 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, the sign is correct, and the characteristic is larger by 128 than the correct characteristic.

5.10.3 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.

5.10.4 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.

5.11 SUPERVISOR CALL INTERRUPTION

Refer to the SVC instruction description in Section 7.1 for the detailed effects of a supervisor call interruption. The supervisor call interruption occurs as a result of the execution of the SUPERVISOR CALL instruction. It causes the current PCW and other information to be stored on the system stack and a new SVC PCW to be constructed. The contents of bit positions 8-15 of the SUPERVISOR CALL instruction become the interruption code of the old PCW on the system stack.

Programming Note: The name "supervisor call" indicates that one of the major purposes of the interruption is the switching from problem to supervisor state. This major purpose does not preclude the use of this interruption for other types of status switching. The interruption code may be used to convey a message.

5.12 MACHINE CHECK INTERRUPTION

The machine check interruption provides a means for reporting to the program the occurrence of machine malfunctions. Information is provided to assist the program in determining the location of the fault.

A 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 the interruption code. An interruption code of 1 indicates a main memory parity error. An interruption code of 2 indicates one of two conditions: either an IOP requested permission from the CP to present an I/O interruption, permission was granted, and then the IOP responded with a request to do something other than present an interruption; or an IOP requested permission to present an I/O interruption, permission was granted, and then the IOP did not respond within a reasonable period of time (time-out failure).

A machine check interruption may be masked off by turning off the machine check interruption mask bit in the PCW. A machine check interruption that is masked off causes entry into Control mode.

The machine check reporting area is filled in as follows, depending on the interruption code:

Code = 1 Bytes 0-3 contain the approximate physical address where the parity error occurred. Bytes 4-5 contain the data as read from memory.

Code = 2 Byte 0 contains the device address passed back during an invalid response to the granting of permission to present an interruption, or contains X'FF' if the IOP did not respond within a reasonable period of time.

Any program or supervisor call interruptions that would have occurred as a result of the current operation are eliminated. Any instruction in progress when a machine check occurs is aborted.

5.12.1 VS100 Machine Checks

Machine checks on the VS100 cause workstation 0 to display the message, "MACHINE CHECK xxx", where "xxx" is one of the following:

001 ECC error on main memory read by CP. The Old Machine Check PCW points to the instruction after the one causing the error. For instructions of RX type only, location X'74' of main memory contains the erroneous data and location X'78' contains its main memory address.

003 Error on main memory write by CP. The Old Machine Check PCW points to the instruction after the one causing the error.

017 Bus transaction log overflow (more than 128 entries). Old Machine Check PCW points to the instruction after the one causing the error.

018 Destination IOP has rejected CP or BA communication. Old Machine Check PCW points to the instruction whose execution followed the error. (Note that the previous instruction may not have caused the error.)

019 Both errors 017 and 018 have occurred.

020 Destination processor has rejected CP communication. Machine Check Old PCW points to the instruction whose execution followed the error. (Note that the previous instruction may not have caused the error.)

021 Both errors 017 and 020 have occurred.

022 Both errors 018 and 020 have occurred.

023 Errors 017, 018, and 020 have occurred.

5.13 PRIORITY OF INTERRUPTIONS

During execution of an instruction, several interruption-causing events may occur simultaneously. The instruction may give rise to a program interruption or a clock interrupt, a machine check may occur, and an I/O interruption request may be made, all at the same time. Instead of the program interruption, a supervisor call interruption might occur; however, both cannot occur since these two interruptions are mutually exclusive. Simultaneous interruption requests are honored in a predetermined order.

Requests for interruption existing concurrently at the end of an instruction are honored in descending order of priority, as follows:

Machine check
Supervisor call
Program
Clock
Input/Output.

The action consists of storing the current PCW in the old PCW and fetching the new PCW belonging to the interruption first taken. This new PCW is subsequently stored without any instruction execution, and the next interruption PCW is fetched. This storing and fetching continues until no more interruptions are to be serviced. Further interrupts are honored if and only if the new PCW has them enabled. The I/O or clock interruptions are taken only if the immediately preceding PCW indicates that the system is interruptible for I/O or clock interruptions. The interruption code of a new PCW is not loaded since it provides no useful information.

Instruction execution is resumed using the last-fetched PCW. The order of executing interruption subroutines is therefore the reverse of the order in which the PCWs are fetched.

Programming Note: The order in which simultaneous interruption requests are honored can be changed to some extent by masking. The priority rule applies to simultaneous interruption requests; an interruption request made after some interruptions have already been taken is honored according to the priority prevailing at the moment of the request.

CHAPTER 6
CONTROL MODE

6.1 INTRODUCTION

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

1. Load group--Contains commands for initializing the operating system, loading a stand-alone program, loading a diagnostic program, or restarting a program (from an initialized state).

2. Debug group--Contains 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, hard copy dump of memory and registers, and virtual address translation.

Control mode uses the lowest-addressed workstation on the first I/O processor (workstation 0) for communication with the machine operator. If this workstation is not powered on, the computer will wait until it is turned on. Control mode uses only the top line (line 1) of the CRT display. The previous contents of the line are saved on entry into Control mode and are restored on exit; therefore, Control mode is transparent to the program that is using this workstation.

6.2 METHODS OF ENTRY

6.2.1 Entry during Program Execution

In this case, the following message is displayed at row 1:

```
CONTROL MODE        W XXXXXXXX XXXXXXXX
```

The PCW displayed (Xs above) is the current PCW and can be examined to determine why Control mode was entered.

If PCW bit 33 (Control mode bit) equals 1, then the program has entered Control mode by loading this PCW (with instruction LPCW, STNSM, STOSM, etc.).

If PCW bit 33 is set to 0, then Control mode was entered by the operator's pressing the Control mode button on the CPU. However, if the words CONTROL MODE are blinking, a machine check has occurred. In the case of a machine check, the interruption code (first byte) of the PCW will indicate which type of machine check has occurred.

When Control mode is entered during program execution, the Debug group of Control mode commands is available.

6.2.2 Entry from an Initialization Procedure

In this case, the following message is displayed at row 1:

```
CONTROL MODE          F 04
```

When Control mode is entered in this manner, both the Load group and the Debug group of Control mode commands are available. Pressing the ENTER key causes a Load command to be issued to the fixed or only volume of an assumed disk device at address 04. If this is not desired, the NEW LINE key will clear the command to allow other commands to be entered.

6.3 INITIALIZATION PROCEDURES

The actions taken when the LOAD button is pressed are a subset of those taken when the processor is powered on. The actions which each of them initiates are as follows:

1. All IOPs are initialized and leave no I/O outstanding.

2. The instruction address field of the current PCW is set to 256 and the other PCW bits are set to 0.

3. Main memory locations 32-41 are set to X'40 000100 0800 000000 00' (default IOCW for Load group commands).

4. Local page table 0 is set for virtual address equal to physical address. The local page frame table is reset (i.e., all PFT entries are set for no reference, no change). Local page tables 1 and 2 are reset (page fault state).

5. Control mode is entered.

Additional results of power-on are that memory is zeroed, the clock is set to 0, and the comparator is set to all 1s.

6.4 CONTROL MODE COMMANDS AND RESPONSES

The ENTER key must be pressed after the bracketed information in Subsections 6.4.1 and 6.4.2 has been supplied. In the following discussion, n represents one hexadecimal digit.

6.4.1 Load Group Commands

[F nn] and [R nn] are Load group commands. These commands allow the Control mode user to perform a complete I/O operation using device nn. They are accepted only when all I/O devices are inactive (through use of the LOAD or INITIALIZE button). The I/O operation to be performed is indicated by the IOCW. The IOCW is located through the IOCA. F and R denote fixed and removable disk volumes, respectively. (This distinction is ignored if a device has only a fixed or only a removable volume.)

Default IOCA, Default IOCW - When a Load command is entered, main memory location 32 and the IOCA for the device are modified as follows:

Location 32 = X'40' or X'41' (READ command in IOCW)

IOCA for device = X'0020' (Load IOCA)

The IOCW set during initialization causes a 2048-byte READ to memory location 256 for a fixed-volume disk drive. For a removable-volume device, the first byte of the IOCW is set to X'41' by the [R nn] command.

A Load group command causes the the CP to do the following:

1. Write the device number (nn) at location 80 (IO old PCW). Create the IOCA and IOCW command byte.

2. Clear all I/O interrupts and issue an SIO instruction to device nn. If the condition code returned is 0, proceed; otherwise, terminate the Load command with a message.

3. Accept all outstanding interruptions and ignore them until an interruption from device nn is received. (In general, there will not be any other interruptions outstanding; however, if present, interruptions from devices other than device nn are ignored and lost.)

4. Inspect the IOSW returned by device nn and return the appropriate message.

The CP then makes one of the following responses to the Load group command:

1. If the Load SIO is completed successfully, Control mode will exit immediately and instruction execution proceeds under control of the current PCW.

2. A message, <u>INV</u> <u>DEV</u>, is returned by a Load command for rejected SIO operations (condition code other than 0). The message indicates that the address (nn) referenced a nonexistent IOP (nonexistent device, condition code 3).

3. The following two messages may be returned after I/O completion (i.e., storing of an IOSW by device nn). Bits 0-1 of the IOSW are examined.

a. INT REQ--Bit 0 = 1--Intervention required indicated.

b. I/O ERR--Bit 1 = 0--Abnormal I/O completion (hard error).

6.4.2 Debug Group Commands

The following commands are useful in debugging a program.

[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 6.

For n = 8, floating-point register 0 is displayed.
For n = A, floating-point register 2 is displayed.
For n = C, floating-point register 4 is displayed.
For n = E, floating-point register 6 is displayed.

Odd values of n (such as 9, B, D) cause the low half (i.e., low-order 32 bits) of one register and the high half of the next register to be displayed. (A value of n = 7 results in display of control register 7 followed by the high half of floating-point register 0. A value of n = F results in display of the low half of floating-point register 6 followed by control register 0.)

[P nnnnnn] Displays eight bytes of physical memory from the physical address given. Non-display indicates an invalid address.

[V nnnnnn] Displays the condition code (high-order byte of R1) and remaining contents of R1 resulting from an LPA instruction. If the translation is successful, eight bytes of memory are also displayed from the given virtual address.

[W] Displays the PCW.

[M nnnnnnnn nnnnnnnn] Modifies eight bytes displayed as a result of one of the five preceding commands.

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

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

[D nnnnnn] [nnnnnn] Causes a dump of the PCW, the registers, and main memory to a printer (device 03). The first physical address entered is rounded down to the nearest 32-byte boundary. Each line has a 3-byte address followed by 32 bytes of memory. Memory contents are printed, beginning at the rounded first address, until the line containing the byte location below the second address is reached, the end of memory is reached, or the operator terminates the dump by striking any workstation key. If the printer is off line, the Dump command remains pending. (The printer may be turned on and the Dump command will continue in this case.) Any pending printer interruption for device 03 is lost when a dump to that printer is requested. If device 03 is not a printer, this command is ineffective.

NOTE

When main memory is modified, the change bit is set in the local page frame table (CP local memory).

In control mode no device other than workstation 0 will be serviced by the Control mode I/O processor (i.e., keystrokes for other workstations on the IOP are ignored).

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

The Dump command will be executed even if an I/O command or I/O interruption was active for the printer (device 03). In this case, the previous printer status is discarded.

6.4.3 Screen Manipulation Keys

The following workstation keys are useful for entering and modifying data.

NEW LINE - This key cancels (and clears) the partially-entered line.

SPACE BAR - During a Modify command, this key retains the previous data at the current cursor position.

ENTER - The key must be pressed after the command letter and numbers (if any). After a Modify command, this key may be pressed before all data has been entered, allowing unmodified data to remain unchanged.

CHAPTER 7
INSTRUCTIONS

7.1 GENERAL INSTRUCTION SET

The following instructions represent the basic instruction set for the VS. In addition to these universal machine instructions, some extended mnemonic codes, such as JSI, are discussed in the chapter on machine instructions in the VS Assembler Language Reference Manual. 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. The short floating-point instructions (available as an option to VS80 systems running at least Version 3.04 microcode, and as a standard item to all VS25 and VS100 systems) are denoted by the word "(optional)" next to their format diagrams.

Note that instructions are ordered alphabetically by name in this chapter, and alphabetically by mnemonic in the index.

ADD (AR, A)

```
AR    R1,R2                    (RR)
 ______________________________
|               | R  | R  |
|      1A       |  1 |  2 |
|_______________|____|____|
 0              8   12   15

A     R1,D2(X2,B2)             (RX)
 ______________________________________________________________
|               | R  | X  | B  |                      |
|      5A       |  1 |  2 |  2 |          D           |
|               |    |    |    |          2           |
|_______________|____|____|____|______________________|
 0              8   12   16   20                     31
```

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)

 ___________________________________________   _____________    ______
|                |  L  |  L  |  B   | / /D |    B    |/ / D  |
|       FA       |  1  |  2  |  1   | - - 1|    2    |- -  2 |
|________________|_____|_____|______|/ /___|_________|/ /____|
0                8    12    16     20     32        36    47
```

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)

        ______________________________
        |               | R  | R  |
        |      3A       |  1 |  2 |
        |_______________|____|____|
        0               8   12   15

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

        _______________________________________________
        |               | R  | X  | B  |       D      |
        |      7A       |  1 |  2 |  2 |       2      |
        |_______________|____|____|____|______________|
        0               8   12   16   20             31
```

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)

```
 _______________________________________________________________
|               | R   | X    | B    |                  D        |
|      4A       |  1  |   2  |   2  |                  2        |
|_______________|_____|______|______|___________________________|
0               8     12     16     20                          31
```

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)

 ______________________________
|              | R   | R    |
|      1E      |  1  |   2  |
|______________|_____|______|
0              8    12   15

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

 ____________________________________________________________
|              | R   | X    | B    |                        |
|      5E      |  1  |   2  |   2  |            2           |
|______________|_____|______|______|________________________|
0              8    12     16     20                       31
```

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)
 ___________________________
|                | | R  | R  |
|      2A        |0|  1 |  2 |
|________________|_|____|____|
 0               8,9  12   15
```

```
AER    R1,R2                    (RR, Short)
 ___________________________
|                | | R  | R  |
|      2A        |1|  1 |  2 |   (optional)
|________________|_|____|____|
 0               8,9  12   15
```

```
AD     R1,D2(X2,B2)             (RX, Long)
 ________________________________________________________
|                | | R  |  X  |  B  |                    |
|      6A        |0|  1 |   2 |   2 |          2         |
|________________|_|____|_____|_____|____________________|
 0               8,9  12     16    20                  31
```

```
AE     R1,D2,(X2,B2)            (RX, Short)
 ________________________________________________________
|                | | R  |  X  |  B  |          D         |
|      6A        |1|  1 |   2 |   2 |          2         |   (optional)
|________________|_|____|_____|_____|____________________|
 0               8,9  12     16    20                  31
```

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)

 ______________________________________________________________
|                  | | R  |  X   |  B   |         D           |
|        6E        |0|  1 |   2  |   2  |          2          |
|__________________|_|____|______|______|_____________________|
0                  8,9   12     16     20                    31
```

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

 ______________________________________________________________
|                  | | R  |  X   |  B   |         D           |
|        6E        |1|  1 |   2  |   2  |          2          |  (optional)
|__________________|_|____|______|______|_____________________|
0                  8,9   12     16     20                    31
```

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)

```
 ______________________________
|               | R    | R     |
|      14       |  1   |  2    |
|_______________|______|_______|
 0              8     12     15
```

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

```
 ______________________________________________________________
|               | R    | X     | B     |                       |
|      54       |  1   |  2    |  2    |           D           |
|               |      |       |       |           2           |
|_______________|______|_______|_______|_______________________|
 0              8     12     15                              31
```

NI D1(B1),I2 (SI)

```
 ______________________________________________________________
|               |       I      | B     |           D           |
|      94       |        2     |  1    |           1           |
|_______________|______________|_______|_______________________|
 0              8             16      20                     31
```

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

```
 ______________________________________________________________
|               |              | B     | / /D |   B    |/ / D  |
|      D4       |       L      |  1    | - - 1|   2    |- -  2 |
|_______________|______________|_______|_/_/__|________|/_/____|
 0              8             16      20     32       36     47
```

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.

BIT RESET (BRESET)

```
BRESET   D1(B1),M2           (SI)
_________________________________________________________________
|               |       M       | B   |            D             |
|      9D       |         2     |  1  |             1            |
|_______________|_______________|_____|__________________________|
0               8               16    20                       31
```

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)

```
 ____________________________________________________________
|                |      M       |  B   |          D          |
|       9C       |      2       |  1   |          1          |
|________________|______________|______|_____________________|
0                8              16     20                   31
```

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)
 ______________________________________________________
|               |      M      |  B  |          D        |
|      9E       |      2      |  1  |          1        |
|_______________|_____________|_____|___________________|
0               8             16    20                  31
```

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)
 _______________________________
|                |  R  |  R  |
|       05       |   1 |   2 |
|________________|_____|_____|
0                8    12    15
```

```
BAL    R1,D2(X2,B2)          (RX)
 ____________________________________________________________
|                |  R  |  X  |  B  |              D              |
|       45       |   1 |   2 |   2 |              2              |
|________________|_____|_____|_____|_____________________________|
0                8    12    16    20                            31
```

BRANCH AND LINK (RELATIVE) (RBAL)

```
RBAL    R1,L2                (RL)
 _______________________________________________________________
|                |  R  |                    L                    |
|       75       |   1 |                    2                    |
|________________|_____|_________________________________________|
0                8    12                                        31
```

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)

 ____________________________________________________________
|                |  M  |  R  |  B  |           D             |
|       99       |   1 |   3 |   2 |           2             |
|________________|_____|_____|_____|_________________________|
0                8     12    16    20                       31
```

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)

 ___________________________________________________________________
|                   |  S  |  X   |  B   |                          |
|        81         |   1 |    2 |    2 |            2             |
|___________________|_____|______|______|__________________________|
 0                  8     12     16     20                        31
```

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

This manual is updated by: Addendum 800-1100PO-04.01

BRANCH AND LINK STACK (RELATIVE) (RBALS)

```
RBALS    R1,L2                (RL)

 ______________________________________________________
|              | R  |                L                 |
|      73      |  1 |                 2                |
|______________|____|__________________________________|
 0             8   12                                31
```

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)

 _____________________________
|              | M  | R  |
|     07       |  1 |  2 |
|______________|____|____|
 0              8   12   15
```

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

 ______________________________________________________________
|              | M  |  X  |  B  |                     |
|     47       |  1 |   2 |   2 |          2          |
|______________|____|_____|_____|_____________________|
 0              8   12    16    20                    31
```

BRANCH ON CONDITION (RELATIVE) (RBC)

```
RBC    M1,L2                 (RL)

 ____________________________________________________________
|              | M  |                L                     |
|     77       |  1 |                2                     |
|______________|____|______________________________________|
 0              8   12                                     31
```

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)
 _______________________________________________________________
|               | M  | X  |                                    |
|      65       |  1 |  2 |                 2                  |
|_______________|____|____|____________________________________|
 0               8   12    16                                 31
```

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

This manual is updated by: Addendum 800-1100PO-04.01

BRANCH ON CONDITION STACK (BCS)

```
BCS    M1,S2                 (RS)
 ____________________________
|               |  M  |  S  |
|      01       |   1 |   2 |
|_______________|_____|_____|
0               8    12    15
```

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.

If the branch is to be taken, the stack is referenced and the branch address is obtained from the stack. If the branch is not taken, the stack is not referenced and no stack violations are detected.

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)
 _______________________________
|              | R   | R    |
|     06       |   1 |   2  |
|______________|_____|______|
 0              8    12   15

BCT     R1,D2(X2,B2)          (RX)
 ______________________________________________________________
|              | R   | X    | B    |                    |
|     46       |   1 |   2  |   2  |          2         |
|______________|_____|______|______|____________________|
 0              8    12     16     20                  31
```

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)
 _____________________________________________________________
|              | R |                     L                    |
|      76      | 1 |                     2                    |
|______________|___|__________________________________________|
0              8  12                                         31
```

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)

```
 _____________________________________________________________
|                | R  | R  | B  |                             |
|       86       |  1 |  3 |  2 |              2              |
|________________|____|____|____|_____________________________|
0                8    12   16   20                           31
```

BRANCH ON INDEX HIGH (RELATIVE) (RBXH)

RBXH   R1,R3,L2   (RRL)

```
 _______________________________________________________________
|                | R  | R  |                 L                 |
|       66       |  1 |  3 |                 2                 |
|________________|____|____|___________________________________|
0                8    12   16                                  31
```

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)

 ___________________________________________________________________
|                 | R  | R  | B  |                                  |
|       87        |  1 |  3 |  2 |                D                 |
|                 |    |    |    |                 2                |
|_________________|____|____|____|__________________________________|
 0                 8   12   16   20                                31
```

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)

```
 ______________________________________________________________
|              | R    | R    |                   L             |
|      67      |  1   |  3   |                    2            |
|______________|______|______|_________________________________|
0              8     12     16                               31
```

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)
 ____________________________
|           | R   | R   |
|    19     |  1  |  2  |
|___________|_____|_____|
0           8    12   15
```

```
C   R1,D2(X2,B2)            (RX)
 ______________________________________________________
|           | R   | X   | B   |                 D     |
|    59     |  1  |  2  |  2  |                  2    |
|___________|_____|_____|_____|_______________________|
0           8    12    16    20                      31
```

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)

 _____________________________
|               | | R | R    |
|      29       |0| 1 | 2    |
|_______________|_|___|______|
0               8,9  12   15

CER    R1,R2                    (RR, Short)

 _____________________________
|               | | R | R    |
|      29       |1| 1 | 2    |  (optional)
|_______________|_|___|______|
0               8,9  12   15

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

 ___________________________________________________________
|               | | R | X    | B    |           D          |
|      69       |0| 1 | 2    | 2    |           2          |
|_______________|_|___|______|______|______________________|
0               8,9  12     16     20                     31

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

 ___________________________________________________________
|               | | R | X    | B    |           D          |
|      69       |1| 1 | 2    | 2    |           2          |  (optional)
|_______________|_|___|______|______|______________________|
0               8,9  12     16     20                     31
```

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)

 ___________________________________________    _________    ______
|                 |  L  |  L  |  B  | / /D |  B  | / /D |
|       F9        |  1  |  2  |  1  |     1|  2  |     2|
|_________________|_____|_____|_____|/_/__|_____|/_/___|
0                 8    12    16    20    32    36    47
```

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.

This manual is updated by: Addendum 800-1100PO-04.01

COMPARE HALFWORD (CH)

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

 _____________________________________________________________
|                 |  R   |  X   |  B   |                      |
|       49        |   1  |   2  |   2  |          2           |
|_________________|______|______|______|______________________|
0                 8      12     16     20                     31
```

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)

 _______________________________
|               | R   | R   |
|      15       |  1  |  2  |
|_______________|_____|_____|
0               8    12   15

CL     R1,D2(X2,B2)            (RX)
 _______________________________________________________________
|               | R   | X   | B   |                           |
|      55       |  1  |  2  |  2  |             2             |
|_______________|_____|_____|_____|___________________________|
0               8    12    16    20                          31

CLI    D1(B1),I2               (SI)
 _______________________________________________________________
|               |           | B   |             D             |
|      95       |     I     |  1  |             1             |
|_______________|___________|_____|___________________________|
0               8          16    20                          31

CLC    D1(L,B1),D2(B2)         (SS)
 _______________________________________________________________
|               |           | B   | / /D | B    |/ / D |
|      D5       |     L     |  1  | - - 1|  2   |- -  2|
|_______________|___________|_____|_/_/__|______|/_/___|
0               8          16    20     32     36     47
```

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)
 ___________________________________________________________
|              |  R  |  M  |  B  |              D           |
|      BD      |  1  |  3  |  2  |              2           |
|______________|_____|_____|_____|__________________________|
0              8    12    16    20                          31
```

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, whether or not the exception for the inaccessible part is indicated is unpredictable. 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)
 ____________________________
|                |  R  |  R  |
|       OF       |  1  |  2  |
|________________|_____|_____|
0                8    12   15
```

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)

```
 ________________________________________   __________    _____
|          | L  | I  | L   | B  |/ /D | B     | / /D  |
|    E5    | 1  |  3 |  2  |  1 |    1|  2    |    2  |
|__________|____|____|_____|____|/ /__|_______|_/ /___|
0          8    16   24    32   36    48      52     63
```

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)

 ___________________________   _________________   ___________
|       | R  | R  | B  |    / /    D    | B  |    / /    D     |
|  F6   |  1 |  2 |  1 |    / /    1    |  2 |    / /    2     |
|_______|____|____|____|___/ /_________|____|___/ /__________|
0       8   12   16   20              32   36               47
```

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)

 ____________________________________
|               | R   |///////|
|      OC       |  1  |///////|
|_______________|_____|///////|
 0               8     12   15
```

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 will return condition code 0, and program execution will continue.

Bits 24 to 31 of R1 identify the device. Bits 0 to 23 are ignored.

The CIO instruction is discussed in greater detail in Chapter 8.

Resulting Condition Code

0 I/O operation accepted or device for which CIO not supported, execution proceeding

1 Device busy with previous operation, or interruption other than IOP NOW READY is pending

2 IOP busy

3 IOP not operational

Program Exceptions

Privileged operation

Programming Note

Telecommunications (TC) IOPs use the SIO instruction instead of CIO for memory diagnostic operations.

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

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

```
 ____________________________________________________________
|                |   R   |   X   |   B   |         D         |
|       7F       |    1  |    2  |    2  |          2        |
|________________|_______|_______|_______|___________________|
 0                8      12      16      20                  31
```

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)

 ________________________________________________________
|               | R   | X   | B   |          D          |
|      7E       |   1 |   2 |   2 |          2          |
|_______________|_____|_____|_____|_____________________|
 0              8    12    16    20                    31
```

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)

```
 ______________________________________________________________
|                 | R   | X   | B   |             D            |
|       4F        |   1 |   2 |   2 |               2          |
|_________________|_____|_____|_____|__________________________|
0                 8    12    16    20                         31
```

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)

4E	R 1	X 2	B 2	D 2
0	8	12	16	20 31

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)
 ______________________________
|              | R    | R     |
|      2F      |   1  |   2   |
|______________|______|_______|
0              8     12   15
```

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)
 _____________________________
|              | R   | R    |
|     2E       |  1  |  2   |
|______________|_____|______|
0              8    12  15
```

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)
________________________________________________________________
|                | R   | X   | B   |                           |
|       51       |  1  |  3  |  3  |             2             |
|                |     |     |     |                           |
________________________________________________________________
0                8    12    16    20                          31
```

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.

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)
 ______________________________________________________
|              | R  |///////| B  |                    |
|      A0      | 1  |///////| 3  |          2         |
|              |    |///////|    |                    |
 ______________________________________________________
0              8   12     16    20                   31
```

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.

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)

 ____________________________________________________________
|             | R  |///////| B  |                    D       |
|     A1      |  1 |///////|  3 |                    2       |
|_____________|____|///////|____|____________________________|
0             8    12      16   20                         31
```

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)
 ______________________________
|              | R  | R   |
|     1D       |  1 |  2  |
|______________|____|_____|
0              8   12   15
```

```
D   R1,D2(X2,B2)              (RX)
 ______________________________________________________________
|              | R  |  X  |  B  |              D              |
|     5D       |  1 |   2 |   2 |              2              |
|______________|____|_____|_____|_____________________________|
0              8   12    16    20                            31
```

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)
 ______________________________
|               | | R  | R    |
|      2D       |0|  1 |  2   |
|_______________|_|____|______|
 0               8,9  12   15

DER    R1,R2                    (RR, Short)
 ______________________________
|               | | R  | R    |
|      2D       |1|  1 |  2   |   (optional)
|_______________|_|____|______|
 0               8,9  12   15

DD     R1,D2(X2,B2)             (RX, Long)
 ______________________________________________________________
|               | | R  | X    | B    |                        |
|      6D       |0|  1 |  2   |  2   |           2            |
|_______________|_|____|______|______|________________________|
 0               8,9  12     16     20                      31

DE     R1,D2(X2,B2)             (RX, Short)
 ______________________________________________________________
|               | | R  | X    | B    |           D            |
|      6D       |1|  1 |  2   |  2   |           2            |   (optional)
|_______________|_|____|______|______|________________________|
 0               8,9  12     16     20                      31
```

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

DIVIDE DECIMAL (DP)

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

 ___________________________________________  __________  ________
|              | L  | L  | B   | / /D | B   |/ / D  |
|      FD      |  1 |  2 |  1  | - - 1|  2  |- -  2 |
|______________|____|____|_____|/_/___|_____|/_/____|
0              8    12   16    20     32    36     47
```

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)
 _________________________________
|               | R  | R  |
|      3D       |  1 |  2 |
|_______________|____|____|
 0               8   12   15

DQ      R1,D2(X2,B2)              (RX)
 ____________________________________________________
|               | R  | X  | B  |         D          |
|      7D       |  1 |  2 |  2 |         2          |
|_______________|____|____|____|____________________|
 0               8   12   16   20                   31
```

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

EDIT (ED)

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

```
 ____________________________________________    ______    ______
|               |               | B    | / /D |  B   |/ / D |
|      DE       |       L       | 1    | - - 1|  2   |- -  2|
|_______________|_______________|______|_/_/__|______|/_/___|
0               8               16     20     32     36    47
```

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 indicates also the negative (on) or positive (off) value of the source, and is used as one factor in the setting of the condition code.

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 a source digit in the high-order part of a byte 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 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 7-1.

Table 7-1. 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 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.

<u>Result Conditions</u>. 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 field edited is nonzero and the significance indicator is on, the condition code is made 1 to indicate a result field less than 0.

When the field edited 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 7-2 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.

Table 7-2. 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		
		0-9	Yes		
	On	0-9	No		
		0-9	Yes		
Message character	Off	**	**	Fill character	Off
	On	**	**	Message character	On

* No effect on result character.

** Not applicable because source digit not examined.

Resulting Condition Code

0 Field is 0
1 Field is less than 0
2 Field is greater than 0
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.

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.

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 was odd) yields the number of bytes in operand 2. However, an I/O operation could overlay operand 1 during instruction execution and invalidate the results of the "scanning operation" just described. If the I/O operation increases the length of operand 2 (by adding digit selectors or significance starters) and causes operand 2 to span a page where previously it did not, then the next higher physical page frame will be accessed. This will cause incorrect (random) data to be used and could also cause an addressing exception if the next higher physical page frame is beyond the limit of main memory.

EDIT AND MARK (EDMK)

```
EDMK    D1(L,B1),D2(B2)         (SS)

 _______________________________________________________________
|               |               |  B   | / /D |  B    |/ / D  |
|      DF       |       L       |  1   | - - 1|  2    |- -  2 |
|_______________|_______________|______|_/_/__|_______|/_/____|
0               8               16     20     32      36     47
```

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
2 Last field is greater than 0
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)

```
 ______________________________________________________________
|                |  R  |  X  |  B  |                           |
|       52       |  1  |  3  |  3  |             2             |
|________________|_____|_____|_____|___________________________|
0                8     12    16    20                        31
```

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 Rl 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.

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)

```
 _____________________________________________________________
|               | R  | X  | B  |                 D            |
|      53       |  1 |  3 |  3 |                 2            |
|_______________|____|____|____|______________________________|
0               8   12   16   20                            31
```

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)
 _____________________________
|               | R  | R  |
|     17        |  1 |  2 |
|_______________|____|____|
 0              8   12   15
```

```
X    R1,D2(X2,B2)              (RX)
 ______________________________________________________________
|               | R  | X   | B   |                        |
|     57        |  1 |  2  |  2  |            2           |
|_______________|____|_____|_____|________________________|
 0              8   12    16    20                       31
```

```
XI    D1(B1),I2                (SI)
 ______________________________________________________________
|               |          | B   |            D           |
|     97        |    I     |  1  |            1           |
|_______________|__________|_____|________________________|
 0              8         16    20                       31
```

```
XC    D1(L,B1),D2(B2)          (SS)
 ______________________________________________________________
|               |          | B   | / /D | B    |/ / D |
|     D7        |    L     |  1  | - - 1|  2   |- -  2|
|_______________|__________|_____|_/_/__|______|/_/___|
 0              8         16    20     32     36     47
```

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)

```
 _______________________________________________________
|             | R   | X   | B   |                       |
|     44      |  1  |  2  |  2  |           2           |
|             |     |     |     |                       |
 -------------------------------------------------------
0             8     12    16    20                    31
```

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)

 _______________________________________   ________________   _______
|        | R   | R   |     B      | / / D     | B     | / / D    |
|   F7   |  1  |  2  |     1      |         1 |  2    |        2 |
|________|_____|_____|____________|_/ /_______|_______|_/ /______|
0        8    12    16           24          32      36        47
```

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)

 _______________________________
|              | R  |///////|
|      03      |  1 |///////|
|______________|____|///////|
0              8    12   15
```

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 24 to 31 of R1 identify the device address. Bits 0 to 23 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 and/or all outstanding interruption conditions pertaining to the device have been cleared. The error completion (EC) and incorrect length (IL) bits may or may not be set in the stored I/O Status Word. HIO clears all interruption conditions that existed at the time the HIO was issued, including IOP NOW READY (and sets condition code 1 in this case).

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.

Resulting Condition Code

0 Device available or HIO not supported
1 Device busy or interruption pending
2 IOP BUSY
3 IOP not operational

Program Exceptions

Privileged operation

Programming Notes

The instruction HALT I/O provides the program with a means of terminating an I/O operation before all data specified in the operation has been transferred or before the operation at the device has reached its normal ending point.

Not all devices support HALT I/O. When it is issued for a device for which it is not supported, the HIO instruction will return condition code 0 or 1 as appropriate, and program execution will continue.

HALVE (FLOATING-POINT) (HDR, HER)

```
HDR    R1,R2                      (RR, Long)

 ____________________________
|              | | R  |  R   |
|      24      |0|  1 |   2  |
|______________|_|____|______|
 0              8,9  12   15
```

```
HER    R1,R2                      (RR, Short)

 ____________________________
|              | | R  |  R   |
|      24      |1|  1 |   2  |   (optional)
|______________|_|____|______|
 0              8,9  12   15
```

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)

```
 ___________________________________________________________
|                |  R  |///////|  B   |                     |
|       A2       |  1  |///////|  3   |          2          |
|                |     |///////|      |                     |
 -----------------------------------------------------------
0                8    12      16     20                    31
```

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.

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)
 ____________________________________________________________
|                  | R   | X   | B   |                       |
|       43         |  1  |  2  |  2  |           2           |
|                  |     |     |     |                       |
 ------------------------------------------------------------
0                  8     12    16    20                     31
```

(In the printed format, the field after B is headed D.)

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)

```
 _______________________________________________________________
|               | R   | M   | B   |              D              |
|      BF       |  1  |  3  |  2  |               2             |
|_______________|_____|_____|_____|_____________________________|
0               8    12    16    20                           31
```

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)

```
 ___________________________________________________________
|              |  M  |  X  |  B  |             D            |
|      61      |   1 |   2 |   2 |             2            |
|______________|_____|_____|_____|__________________________|
 0             8    12    16    20                        31
```

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 pushed onto the system stack, the contents of the three low-order bytes of control register 1 are then pushed onto the stack, preceded by a byte containing binary 0s, and then the contents of general registers 14 to 0 are pushed onto the stack. 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.

The three low-order bytes of the word at the location designated by the second operand are used as the branch address. The second operand must be fullword aligned.

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 if the branch is taken; 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 the 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.

LOAD (LR, L)

```
LR   R1,R2                   (RR)

 ________________________________
|               | R    | R     |
|      18       |  1   |   2   |
|_______________|______|_______|
0               8     12    15
```

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

 _____________________________________________________________
|               | R    | X     | B     |           D          |
|      58       |  1   |   2   |   2   |           2          |
|_______________|______|_______|_______|______________________|
0               8     12      16      20                     31
```

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)

 ____________________________________
|                 | | R  |  R   |
|       28        |0|  1 |   2  |
|_________________|_|____|______|
 0                8,9  12   15

LER    R1,R2                      (RR, Short)
 ____________________________________
|                 | | R  |  R   |
|       28        |1|  1 |   2  |   (optional)
|_________________|_|____|______|
 0                8,9  12   15

LD     R1,D2(X2,B2)               (RX, Long)
 ___________________________________________________________________________
|                 | | R  |   X   |   B   |                D                |
|       68        |0|  1 |    2  |    2  |                 2               |
|_________________|_|____|_______|_______|_________________________________|
 0                8,9  12       16      20                                31

LE     R1,D2(X2,B2)               (RX, Short)
 ___________________________________________________________________________
|                 | | R  |   X   |   B   |                D                |
|       68        |1|  1 |    2  |    2  |                 2               |   (optional)
|_________________|_|____|_______|_______|_________________________________|
 0                8,9  12       16      20                                31
```

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)
 ____________________________________________________________
|              | R    | X    | B    |                        |
|      41      |   1  |   2  |   2  |           2            |
|______________|______|______|______|________________________|
0              8      12     16     20                      31
```

LOAD ADDRESS (RELATIVE) (RLA)

```
RLA   R1,L2                    (RL)
 ____________________________________________________________
|              | R    |                 L                    |
|      71      |  1   |                   2                  |
|______________|______|______________________________________|
0              8      12                                    31
```

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)
 ____________________________
|              |  R  |  R  |
|      12      |  1  |  2  |
|______________|_____|_____|
 0              8    12   15

LT    R1,D2(X2,B2)           (RX)
 ________________________________________________________
|              |  R  |  X  |  B  |           D          |
|      4D      |  1  |  2  |  2  |           2          |
|______________|_____|_____|_____|______________________|
 0              8    12    16    20                     31
```

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)

 ________________________________
|                 | | R | R     |
|       22        |0| 1 |  2    |
|_________________|_|___|_______|
 0                 8,9  12   15

LTER   R1,R2                     (RR, Short)

 ________________________________
|                 | | R | R     |
|       22        |1| 1 |  2    |   (optional)
|_________________|_|___|_______|
 0                 8,9  12   15
```

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)

```
 ______________________________________________________________
|              |  R  |  X  |  B  |               D             |
|      62      |  1  |  2  |  2  |               2             |
|              |     |     |     |                             |
|______________|_____|_____|_____|_____________________________|
0              8    12    16    20                            31
```

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)

   ________________________________
  |              | R    | R    |
  |      13      |   1  |   2  |
  |______________|______|______|
   0              8     12    15
```

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)
     ___________________________________
     |              | | R  |  R  |
     |      23      |0|  1 |   2 |
     |______________|_|____|_____|
      0              8 9   12   15

     LCER    R1,R2                         (RR, Short)
     ___________________________________
     |              | | R  |  R  |
     |      23      |1|  1 |   2 |  (optional)
     |______________|_|____|_____|
      0              8 9   12   15
```

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)

```
 ___________________________________________________________________
|                  | R  | R  | B  |                                |
|        B7        |  1 |  3 |  2 |               D                |
|                  |    |    |    |               2                |
|__________________|____|____|____|________________________________|
0                  8    12   16   20                              31
```

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)
```

	R	X	B	D	
48	1	2	2	2	
0	8	12	16	20	31

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)

This manual is updated by: Addendum 800-1100PO-04.01

LOAD MULTIPLE (LM)

```
LM    R1,R3,D2(B2)               (RS)

 ____________________________________________________________
|                 | R  | R  | B  |                          |
|       98        |  1 |  3 |  2 |            D             |
|                 |    |    |    |            2             |
|_________________|____|____|____|__________________________|
0                 8   12   16   20                         31
```

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)

 ____________________________________
|                |  R     |  R      |
|       11       |    1   |    2    |
|________________|________|_________|
 0               8       12    15
```

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)
     __________________________________
     |              | | R  | R     |
     |     21       |0|  1 |   2   |
     |______________|_|____|_______|
      0              8,9  12    15

     LNER    R1,R2                     (RR, Short)
     __________________________________
     |              | | R  | R     |
     |     21       |1|  1 |   2   |   (optional)
     |______________|_|____|_______|
      0              8,9  12    15
```

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

This manual is updated by: Addendum 800-1100PO-04.01

LOAD OR TRAP (LOT)

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

```
 ______________________________________________________________
|                 |  R  |  X  |  B  |            D             |
|       A8        |   1 |   2 |   2 |            2             |
|_________________|_____|_____|_____|__________________________|
0                 8    12    16    20                         31
```

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 PAGE TABLE (LPT0, LPT1, LPT2)^{P}

```
LPT0    R1,D2(B2)               (RS)

 ________________________________________________________________
|                |  R   |//////|  B   |                          |
|       A3       |   1  |//////|   2  |            D             |
|                |      |//////|      |             2            |
 ----------------------------------------------------------------
 0               8     12     16     20                         31

LPT1    R1,D2(B2)               (RS)

 ________________________________________________________________
|                |  R   |//////|  B   |                          |
|       A4       |   1  |//////|   2  |            D             |
|                |      |//////|      |             2            |
 ----------------------------------------------------------------
 0               8     12     16     20                         31

LPT2    R1,D2(B2)               (RS)

 ________________________________________________________________
|                |  R   |//////|  B   |                          |
|       A5       |   1  |//////|   2  |            D             |
|                |      |//////|      |             2            |
 ----------------------------------------------------------------
 0               8     12     16     20                         31
```

The first operand register contains the address of a page table in main memory in its three low-order bytes, and the length of the page table (in bytes) minus 1 in its high-order byte. The second operand is the address of a byte in main memory that is expected to contain the length minus 1 of the page table currently loaded into local storage for the specified segment. This byte is replaced by the high-order byte of the operand 1 register.

The page table specified by the first operand is copied into the local page table specified by the operation code. LPT0 loads the segment 0 page table, LPT1 the segment 1 page table, and LPT2 the segment 2 table. Bytes of the local page table beyond the length specified in the first operand register but not beyond the length specified by the second operand are set to 0. The first operand page table address must be doubleword aligned, and the new and previous lengths must be within the allowed length for the segment and must be even (length byte in high-order part of register R1 odd), or a specification exception occurs and the instruction is suppressed.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Specification

Privileged operation

Access (fetch, page table addressed by operand 1 register; fetch and store, operand 2)

LOAD PARTIAL PAGE TABLE (LPPT)

```
LPPT    R1,I3,D2(B2)          (RS)

 ________________________________________________________
|              | R    | I    | B    |          D          |
|      A7      |  1   |  3   |  2   |          2          |
|              |      |      |      |                     |
 --------------------------------------------------------
0              8     12     16     20                    31
```

The first operand register contains the address of a page table in main memory in bits 8-31, and the length of the page table (in bytes) minus 1 in bits 0-7. The third operand (I3) indicates the segment number. The second operand indicates the displacement into the local page table in page units.

The partial page table specified by the first operand is copied into the local page table specified by I3. The length and starting displacement are used as described for the previous instruction

A specification exception occurs, and the instruction is suppressed, if any of the following occurs:

- The first operand is not doubleword aligned.
- A segment other than 0, 1, or 2 is indicated by I3.
- The offset and length of the partial page table exceeds the local page table length.
- The offset (operand 2) is odd.
- The length (bits 0-7 of register R1) is odd.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Specification
Privileged operation
Access (fetch, operand 1)

LOAD PCW (LPCW)^{p}

```
LPCW   D1(B1)                 (S)
 _______________________________________________________________
|               |///////////////|   B   |                       |
|      82       |///////////////|   1   |           1           |
|_______________|///////////////|_______|_______________________|
0               8               16      20                     31
```

The two words at the location designated by the operand address replace the PCW. The operand must be word aligned or the instruction will 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)

This manual is updated by: Addendum 800-1100PO-04.01

LOAD PHYSICAL ADDRESS (LPA)^{P}

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

 ____________________________________________________________
|                 | R   | X   | B   |                       |
|       B1        |   1 |   2 |   2 |          2            |
|_________________|_____|_____|_____|_______________________|
0                 8    12    16    20                      31
```

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 by means of the address translation facility. The translation is performed using the contents of the main memory page tables, located through the SCRs.* 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. When the page table entry lies within the specified table length, the segment is valid and the page is not marked with a page fault indication, i.e., its T-RAM entry is not X'8000'. The corresponding reference bit in the local page frame table entry for a successfully translated address is set.

When there would be a segment fault, condition code 1 is set. When the page table entry is X'8000', condition code 2 is set. Whenever the resulting condition code is not 0, the general register designated by the R1 field is zeroed.

Resulting Condition Code

0 Translation available
1 Segment index invalid (segment fault)
2 Page table entry invalid (page fault)
3 Page table length violation (past the end of the page table)

Program Exceptions

Privileged operation

* The VS80 uses local page tables, a 19-bit physical address, and bit positions 13-31 of the general register specified by R1.

LOAD POSITIVE (LPR)

LPR R1,R2 (RR)

10	R_1	R_2
0	8	12 15

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

LOAD POSITIVE (FLOATING-POINT) (LPDR, LPER)

```
LPDR    R1,R2                      (RR, Long)
 ______________________________
|               | | R  |  R    |
|      20       |0|  1 |   2   |
|_______________|_|____|_______|
 0               8,9  12   15

LPER    R1,R2                      (RR, Short)
 ______________________________
|               | | R  |  R    |
|      20       |1|  1 |   2   |   (optional)
|_______________|_|____|_______|
 0               8,9  12   15
```

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)
 ______________________________
|              | | R  |  R    |
|      25      |1|  1 |   2   |  (optional)
|______________|_|____|_______|
0               8,9  12   15
```

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

This manual is updated by: Addendum 800-1100PO-04.01

LOAD SEGMENT CONTROL REGISTER (LSCTL)^{P}

LSCTL R1,R3,D2(X2,B2) (RS)

A3	R1	R3	B2	D2	
0	8	12	16	20	31

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.

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.

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. R1 and R3 must fall in the range 0-7, and 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)
     ___________________________
     |             | | R  |  R  |
     |      25     |0|  1 |  2  |  (optional)
     |_____________|_|____|_____|
      0             8,9  12   15
```

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

LOAD SPECIAL REGISTER (LSREG)^{P}

```
LSREG   D1(B1)                (S)

---------------------------------------------------------
|             |             | B    |                    |
|     9B      |     01      |  1   |     1              |
|             |             |      |                    |
---------------------------------------------------------
0             8             16     20                 31
```

Data is moved from memory to a special 32-bit register, which may be accessed only by the LSREG and STSREG instructions (and by STDD).

The fullword at the address specified by the B1 and D1 fields is moved to the special register. The address must be fullword aligned.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Privileged operation
Access (fetch, operand 1)
Specification

<u>MOVE</u> (MVI, MVC)

```
     MVI    D1(B1),I2                 (SI)
     ________________________________________________________
     |           |         I    |  B   |          D          |
     |    92     |         2    |  1   |          1          |
     |___________|______________|______|_____________________|
      0           8             16    20                    31

     MVC    D1(L,B1),D2(B2)           (SS)
     ________________________________________________________
     |           |              |  B   | / /D |  B   |/ / D  |
     |    D2     |      L       |  1   | - - 1|  2   |- -  2 |
     |___________|______________|______|_/_/__|______|/_/____|
      0           8             16    20     32     36     47
```

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.

<u>Resulting Condition Code</u>

The condition code remains unchanged.

<u>Program Exceptions</u>

Access (fetch, operand 2 of MVC; store operand 1, MVI and MVC)

<u>Programming Note</u>

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)

 ____________________________
|             | R   | R   |
|     0E      | 1   | 2   |
|_____________|_____|_____|
0             8    12   15
```

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)

 ____________________________________________________   ___________   _________
|               |              |  B    | / /D |  B     |/ / D  |
|      D1       |      L       |   1   | - - 1|   2    |- -  2 |
|_______________|______________|_______|/_/___|________|/_/____|
 0              8              16      20       32     36     47
```

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)

 _______________________________________   _________   ______
|               | L  | L  | B   | / /D |  B   |/ / D |
|      F1       | 1  | 2  | 1   | - - 1|  2   |- -  2|
|_______________|____|____|_____|_/_/__|______|/_/___|
0               8   12   16    20     32     36    47
```

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)

```
 _________________________________________   _____________   _____________
|     | L   | I   | L   | B   | / / D | B | / / D       |
| E2  |  1  |  3  |  2  |  1  | - -  1|  2| - -  2      |
|_____|_____|_____|_____|_____|/ /____|___|/ /__________|
0     8    16    24    32    36      48  52            63
```

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)

_______________________________________________    __________    _______
|            |            | B     | / /D |  B      |/ / D  |
|     D3     |     L      |  1    | - - 1|  2      |- -  2 |
|            |            |       |_/_/__|_________|/_/____|
0            8            16     20     32       36     47
```

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)
 _____________________________
|            | R  | R  |
|     1C     |  1 |  2 |
|____________|____|____|
0            8   12   15
```

```
M    R1,D2(X2,B2)           (RX)
 ____________________________________________________
|            | R  | X  | B  |                        |
|     5C     |  1 |  2 |  2 |           2            |
|____________|____|____|____|________________________|
0            8   12   16   20                       31
```

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)

 ______________________________________
|               | |  R  |   R   |
|      2C       |0|  1  |   2   |
|_______________|_|_____|_______|
0              8,9    12     15

MER    R1,R2                      (RR, Short)

 ______________________________________
|               | |  R  |   R   |
|      2C       |1|  1  |   2   |    (optional)
|_______________|_|_____|_______|
0              8,9    12     15

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

 _________________________________________________________________
|               | |  R  |   X   |   B   |            D             |
|      6C       |0|  1  |   2   |   2   |            2             |
|_______________|_|_____|_______|_______|__________________________|
0              8,9    12      16      20                        31

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

 _________________________________________________________________
|               | |  R  |   X   |   B   |            D             |
|      6C       |1|  1  |   2   |   2   |            2             |   (optional)
|_______________|_|_____|_______|_______|__________________________|
0              8,9    12      16      20                        31
```

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 (for MER and ME, 7 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 (for MER and ME, all 7 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 (MD and ME only)
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.

This manual is updated by: Addendum 800-1100PO-04.01

MULTIPLY DECIMAL (FLOATING-POINT) (MQR, MQ)

```
MQR     R1,R2                  (RR)
 ________________________________
|              | R   | R   |
|      3C      |   1 |   2 |
|______________|_____|_____|
0              8    12    15

MQ     R1,D2(X2,B2)            (RX)
 ____________________________________________________
|              | R   | X   | B   |           D          |
|      7C      |   1 |   2 |   2 |           2          |
|______________|_____|_____|_____|______________________|
0              8    12    16    20                     31
```

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)

 ________________________________________________________
|                  | R  | X  | B  |                      |
|        4C        |  1 |  2 |  2 |          2           |
|__________________|____|____|____|______________________|
0                  8   12   16   20                     31
```

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, the exception of 47 bits being 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)

 ______________________________________
|                  | R    | R    |
|        16        |  1   |  2   |
|__________________|______|______|
 0                 8     12       15

O   R1,D2(X2,B2)                  (RX)
 ____________________________________________________________________________
|                  | R    | X    | B    |                                    |
|        56        |  1   |  2   |  2   |                 2                  |
|__________________|______|______|______|____________________________________|
 0                 8     12     16     20                                   31

OI    D1(B1),I2                   (SI)
 ____________________________________________________________________________
|                  |      I      | B    |                 D                  |
|        96        |      2      |  1   |                 1                  |
|__________________|_____________|______|____________________________________|
 0                 8            16     20                                   31

OC    D1(L,B1),D2(B2)             (SS)
 __________________________________________   ___________   ___________
|                  |             | B    | / /D | B      | / /D   |
|        D6        |      L      |  1   |    1|  2      |     2  |
|__________________|_____________|______|_/_/__|________|_/_/____|
 0                 8            16     20     32       36       47
```

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 0s in all positions corresponding to the first operand bits to be set to 0.

PACK (PACK)

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

```
 ____________________________________________ ______________ _____________
|                    | L   | L   | B   | / /D |  B    |/ / D  |
|        F2          | 1   | 2   | 1   | - - 1|  2    |- - 2  |
|____________________|_____|_____|_____|_/_/__|_______|/_/____|
 0                    8    12    16    20     32      36     47
```

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.

PACK AND ALIGN (PAL)

```
PAL   D1(L1,B1),D2(L2,B2)      (SS)

 ___________________________________________ ___________ _________
|               | L  | L  | B  | / /D | B    |/ / D |
|      C4       | 1  | 2  | 1  |- - 1|  2   |- -  2 |
|_______________|____|____|____|_/_/__|______|/_/___|
0               8   12   16   20     32     36     47
```

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 7-3 summarizes the scan order, disregarding any leading or trailing blanks.

Table 7-3. 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:

```
 _______________________________________________________
|///////| Left  |   |   |     2's complement           |
|///////| count | S | D |        of right              |
|///////|_______|___|___|_________count________________|
         16   23 24  25  26                          31
```

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)
 ______________________________
|              | S    | R     |
|      08      |   1  |   2   |
|______________|______|_______|
0              8     12   15
```

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)

```
 ____________________________________________________   ______________
|              |            |  B  |                 |  S  |/ /        |
|      D8      |     L      |  1  |        1        |  2  |- -        |
|______________|____________|_____|_________________|_____|/_/________|
0              8           16    20                32    36          47
```

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)

 ______________________________
|             | S   | R   |
|     09      |  1  |  2  |
|_____________|_____|_____|
0             8    12   15
```

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 bytes that were 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)
     ______________________________________________________
     |              | S  | R  | R  |//////////////////////|
     |      A6      |  1 |  3 |  2 |//////////////////////|
     |              |    |    |    |//////////////////////|
     ------------------------------------------------------
      0             8   12   16   20                    31
```

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)

```
 ______________________________________________________________
|                |  S  |  X  |  B  |            D             |
|       84       |  1  |  2  |  2  |            2             |
|________________|_____|_____|_____|__________________________|
 0               8    12    16    20                        31
```

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)

0B	S1	R2
0	8	12 15

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)

```
 ________________________________________________________________
|                | S  | X  | B  |               D               |
|       BO       |  1 |  2 |  2 |               2               |
|                |    |    |    |                               |
 ----------------------------------------------------------------
0                8    12   16   20                            31
```

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)

 ______________________________________________________________
|          | R   |                      L                      |
|    72    |  1  |                       2                     |
|__________|_____|_____________________________________________|
0          8    12                                           31
```

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)

______________________________________________   _________   _____
|              |              |  S    |// ///|   B    |/ / D   |
|      D9      |      L       |  1    |// ///|   2    |- -  2  |
|              |              |_______|// ///|________|/ /_____|
0              8              16     20     32       36    47
```

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)
_______________________________________________________________
|               | S   | R   | R   |//////////////////////////|
|      A9       |  1  |  3  |  2  |//////////////////////////|
|               |     |     |     |//////////////////////////|
---------------------------------------------------------------
 0               8    12    16    20                        31
```

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)

```
 ________________________________________________________________
|               |  S   |  X   |  B   |                          |
|      85       |   1  |   2  |   2  |             2            |
|_______________|______|______|______|__________________________|
0               8     12     16     20                         31
```

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)

 ______________________________________________________________
|                 |        M        |   B    |        D        |
|       9F        |        2        |   1    |        1        |
|_________________|_________________|________|_________________|
0                 8                 16       20                31
```

The reference and change table (RCT) (for all VS processors) or the T-RAM monitor area (for the VS25 and VS100) is examined or modified with this instruction, according to the value of M2. When used with the RCT, operand 1 must be a physical address; when used with the T-RAM monitor area, it must be a virtual address. The operation of the instruction is explained below.

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. (Refer also to the discussion of paging in Chapter 4 of this manual.) The RCT is an area of local CP memory containing 8K entries of 2 bits each. These are a reference bit and a change bit for each of the up to 8K addressable pages 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 system paging task uses the reference and change bits along with an aging count in deciding which virtual pages to overwrite with new ones during paging operations.

RRCB Reference and Change Table Function

The privileged RRCB instruction allows inspection and modification of the RCT through use of a physical address. The immediate mask (M2) of the instruction, illustrated below, specifies this action.

```
      ___________________________
      | S| R|//|//|//|//|//|//|
      | E| C|//|//|//|//|//|//|
      |_L|_T|//|//|//|//|//|//|
bits   8                      15
```

Figure 1. Format of Operand M2 in RRCB Instruction for Reference and Change Table

The SEL bit selects reference and change bit control (SEL=0) or monitor area control (SEL=1). When SEL=0, RCT=1 causes resetting of the reference and change bits; RCT=0 only reports the current setting of the bits.

The condition code is set to reflect the state of the reference and change bits before they are conditionally reset.

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

MONITOR AREA

The monitor area is a section of local CP memory that is typically used to record loaded T-RAM entries and thereby control user address space. For example, the VS25 and VS100 use it for efficient clearing of a task's T-RAM at the end of each time slice. (Refer also to the description of address translation in Chapter 4 of this manual.) For the VS25 the monitor area comprises 64 entries; for the VS100 it comprises 128 entries.

Monitoring is enabled for a segment of a user's virtual memory if the M bit is set in the SCR for the segment; refer to Figure 4-5 for an illustration of SCR format. During the successful servicing of a T-RAM fault for a virtual page in a segment for which monitoring is enabled, the virtual page address is recorded in the next available monitor area location. At the end of a user's time slice, only those T-RAM locations identified by monitor entries are cleared (i.e., their high order bit is set to 1), rather than the entire T-RAM. Because only a fraction of each segment's possible T-RAM entries are likely to be loaded during a given time slice, clearing the T-RAM using the monitor is much more efficient than clearing the entire T-RAM, which would require individually accessing each of 512 entries per segment.

RRCB Monitor Area Function

The privileged RRCB instruction allows clearing of all or part of the VS25 or VS100 T-RAM monitor area through use of a virtual address. The immediate mask (M2) of the instruction is interpreted differently for the VS25 and VS100, and specifies monitor action as follows:

```
      | S| R|//| A| S| O| M| D|
      | E| C|//| L| E| N| O| M|
      | L| T|//| L| G| E| N| P|
bits    8                    15
```

Figure 2. Format of Operand M2 in RRCB Instruction for VS25 and VS100 T-RAM Monitor Area

The SEL bit selects reference and change bit control (SEL=0) or monitor control (SEL=1). For SEL=1, the remaining bits have the following meaning when set to 1:

RCT - ignored

ALL - Clears all T-RAM entries and monitor entries; used for diagnostics and initialization.

SEG - Clears T-RAM entries for the segment indicated in virtual address operand (operand 1) of the RRCB instruction. For the VS25, if monitoring is enabled, all T-RAM entries recorded by the monitor for any segment are cleared, along with the monitor entries themselves; i.e., SEG=MON. For the VS100, if monitoring is enabled for the segment, only those T-RAM entries recorded for the segment by the monitor are cleared, along with the monitor entries themselves.

ONE - Clears the one T-RAM entry specified by the virtual address operand of the RRCB instruction. (For the VS100 only: also clears the corresponding monitor entry if monitoring is in effect for the segment.)

MON - Clears T-RAM entries recorded in the monitor area, along with the monitor entries themselves.

DMP - Copies (dumps) the entire monitor area to the word-aligned main memory location specified by the virtual address operand of the RRCB instruction; the T-RAM and monitor are left unchanged. (Not currently supported on the VS25.)

For VS100 only: ALL, SEG, and MON functions place a count value of cleared T-RAM entries in general register 0.

Program Exceptions

Access (addressing only, operand 1)
Privileged operation

RETURN AND POP ON CONDITION (RPC)

```
    RPC    M1,R2                     (RR)

    ____________________________________
    |               | M    |  R    |
    |     26        |  1   |   2   |
    |_______________|______|_______|
    0               8     12     15
```

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)

RETURN ON CONDITION (RTC)

```
RTC    M1                     (RR)

 ____________________________________
|               |  M   |///////|
|      04       |   1  |///////|
|_______________|______|///////|
0               8     12     15
```

If the state of the condition code is as specified by M1, this instruction will set general register 15 (the stack pointer) equal to the current value in control register 1, plus 4 (thus skipping over the register 0 save-area contents stored by a JSCI instruction); the instruction will then pop off the system stack general registers 1 to 14. The high byte of control register 1 is set to 0. The next word will then be popped off the stack and its value will replace the current contents of control register 1. One more word will then be popped off the stack and will be used as the program mask byte and address portion of the current PCW. Control will then pass to the address specified in the address portion of the PCW.

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)

SAVE THEN 'AND' SYSTEM MASK (STNSM)^p

STNSM   R1,R3,I2   (RS)

```
 ________________________________________________________________
|                  | R    | R    |                               |
|       AC         |   1  |   3  |               2               |
|__________________|______|______|_______________________________|
 0                 8      12     16                             31
```

This instruction tests whether to save the current PCW status field. If the Rl field of the instruction is 0, the saving is bypassed. If Rl is not 0, the current PCW status field is saved in bits 16-31 of register Rl. Bits 0-15 of Rl 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)

```
 ______________________________________________________________
|                 |  R   |  R   |                              |
|       AD        |   1  |   3  |              2               |
|_________________|______|______|______________________________|
0                 8      12     16                            31
```

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)

```
 ____________________________________________________________
|                  | R      | M     | B     |        D          |
|        B8        |   1    |   3   |   2   |          2        |
|__________________|________|_______|_______|___________________|
0                  8        12      16      20                 31
```

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 result is used. 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.

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 instructon 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

The second operand of SCAN FOR BYTE must not be a literal expression.

For general notes on interruptible instructions, refer to MOVE CHARACTERS LONG.

SET PROGRAM MASK (SPM)

```
SPM   R1                          (RR)
 ______________________________
|              | R   |///////|
|      0D      |  1  |///////|
|______________|_____|///////|
 0              8     12   15
```

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)

```
 ________________________________________________   ___________   ____________
|                     |  L  |  I  |  B  | / /D |  B  |/ / D |
|         F0          |  1  |  3  |  1  |     1|  2  |     2 |
|_____________________|_____|_____|_____|_/_/__|_____|/_/___|
0                     8    12    16    20    32    36    47
```

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)
 ________________________________________________________________
|                |   R   |///////|   B   |                       |
|       8F       |   1   |///////|   2   |           D           |
|                |       |///////|       |           2           |
|________________|_______|///////|_______|_______________________|
 0                8      12      16      20                     31
```

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)

```
 ______________________________________________________
|               | R   |///////| B   |                  |
|      8D       |   1 |///////|   2 |         2        |
|               |     |///////|     |                  |
 ------------------------------------------------------
0               8    12      16    20                31
```

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)

 _______________________________________________________________
|                  | R  |///////| B   |                         |
|        8B        |  1 |///////|  2  |            2            |
|__________________|____|///////|_____|_________________________|
 0                  8    12      16    20                      31
```

(The D label above the 2 in the last field:)

```
                                              D
```

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)

```
 ________________________________________________________________
|                  | R    |///////|  B    |            D         |
|        89        |  1   |///////|   2   |             2        |
|__________________|______|///////|_______|______________________|
 0                  8      12      16      20                   31
```

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)

 ____________________________________________________________________
|                  | R    |///////| B     |                          |
|       8E         |  1   |///////|  2    |            2             |
|                  |      |///////|       |                          |
 --------------------------------------------------------------------
 0                 8     12      16      20                         31
```

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)
 ______________________________________________________________
|              | R   |///////| B   |              D              |
|      8C      |  1  |///////|  2  |              2              |
|              |     |///////|     |                             |
 --------------------------------------------------------------
0              8    12      16    20                            31
```

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)
_____________________________________________________________
|               | R   |///////| B    |          D          |
|      8A       |  1  |///////|  2   |          2          |
|_______________|_____|///////|______|_____________________|
0               8    12      16     20                    31
```

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)
 _______________________________________________________
|              | R  |///////| B  |                      |
|      88      |  1 |///////|  2 |          2           |
|______________|____|///////|____|______________________|
0              8    12      16   20                    31
```

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)

 ______________________________________
|                |  R   |///////|
|       02       |   1  |///////|
|________________|______|///////|
 0                8      12    15
```

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.

The register specified by R1 contains the I/O device address to which the instruction applies. The IOCA must be stored in the appropriate IOCA area before the SIO is issued, and must not be changed until the completion interrupt is accepted.

In the register specified by R1, bit positions 24-31 contain the device address. Bits 0 to 23 are ignored.

```
 _____________________________________
|///////////////////|                 |
|///////////////////|  device         |
|///////////////////| address         |
|///////////////////|_________________|
 0                23 24             31
```

The I/O operation specified by START I/O is initiated if the addressed I/O device and its I/O processor (IOP) are available. The I/O operation is not initiated in the following cases: when the addressed IOP is not connected or is otherwise not operational, when the IOP is unable to service the request (IOP BUSY), or when the device is busy with a previous SIO or CIO or has an interruption pending other than IOP NOW READY. In these cases the instruction is completed and a condition code of 3, 2, or 1, respectively, is returned.

Whenever an SIO instruction is completed with a condition code of 0, the I/O operation has been accepted and a pending I/O interruption will be established on completion of the operation. Until the completion interrupt has been received, the IOCW and IOCA must not be changed. The IOCA and IOCW are not changed by the IOP.

If the addressed IOP is not connected or is otherwise not operational, the I/O operation is not initiated and a condition code of 3 is returned. It is not recommended that this feature be used in a time-critical program; it may take the CP some time for the "time out" to determine that the IOP is not present. If the IOP address is valid but the addressed device is not attached, the SIO is accepted and the IOSW stored on the completion interrupt indicates DEVICE NOT READY.

If the IOP cannot respond to the SIO request, the SIO will complete with a condition code of 2 and the I/O operation will not be started. This condition indicates that an IOP NOW READY interrupt bit will be set in the next IOSW from that IOP. Note that this interrupt may or may not occur on the device on which the IOP BUSY indication was received.

Resulting Condition Code

0 I/O operation accepted, execution proceeding

1 Device busy with previous operation or interruption other than IOP NOW READY pending

2 IOP BUSY

3 IOP not operational

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)

```
 _____________________________________________________________
|              | R    | X    | B    |           D            |
|      50      |  1   |   2  |   2  |            2           |
|______________|______|______|______|________________________|
 0              8      12     16     20                     31
```

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)

```
 ______________________________________________________________
|                | R    | X    | B    |                 D        |
|       42       |   1  |   2  |   2  |                 2        |
|                |      |      |      |                          |
 --------------------------------------------------------------
0                8      12     16     20                        31
```

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)

 _____________________________________________________________
|              |  R   |  M   |  B   |                         |
|      BE      |   1  |   3  |   2  |            2            |
|______________|______|______|______|_________________________|
0              8     12     16     20                        31
```

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)

```
 _______________________________________________________________
|                 | R   | R   | B   |                           |
|       B6        |   1 |   3 |   2 |             2             |
|                 |     |     |     |                           |
 ---------------------------------------------------------------
0                  8    12    16    20                        31
```

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 CP TYPE AND MICROCODE VERSION (STCPID)

```
STCPID R1                          (S)

 ________________________________________________________________
|            |            | R |////////////////////////|
|     9B     |     80     | 1 |////////////////////////|
|____________|____________|___|////////////////////////|
0            8            16  20                      31
```

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 VS80, 3; for the VS100, 4; and for the VS25, 5.

Resulting Condition Code

For the current VS processor, the condition code is always set to 0.

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)

9B	00	B 1	D 1
0	8	16	20 31

Diagnostic information, including the contents of the local page tables and local page frame table, 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. Floating-point registers, control registers, and general registers are stored with high-order and low-order halfwords reversed. The local page frame table is stored in 4-bit entries. The low-order bit (bit 3) is the change bit (0 = set, 1 = clear); bit 2 is the reference bit (0 = set, 1 = clear). The other two bits of each entry are unused.

Data Item (and Decimal Size)	Offset, in Hexadecimal, from Operand 1 Address
File registers (64)	X'0'
Floating-point registers (32)	X'40'
Control registers (32)	X'60'
Auxiliary registers (64)	X'80'
General registers (64)	X'C0'
Page table 0 (128)	X'100'
Page frame table (128)	X'180'
Page table 1 (256)	X'200'
Page table 2 (256)	X'300'

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Access (store, operand 1)
Specification
Privileged operation

Extended Operation Codes

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 (FLOATING-POINT) (STD, STE)

```
STD   R1,D2(X2,B2)            (RX, Long)

 ________________________________________________________________
|                  | | R  |   X   |  B   |                       |
|       60         |0|  1 |    2  |   2  |          2            |
|__________________|_|____|_______|______|_______________________|
 0                 8 9   12      16     20                      31
```

```
STE   R1,D2(X2,B2)            (RX, Short)

 ________________________________________________________________
|                  | | R  |   X   |  B   |           D           |
|       60         |1|  1 |    2  |   2  |          2            |  (optional)
|__________________|_|____|_______|______|_______________________|
 0                 8 9   12      16     20                      31
```

The first operand is stored at the second operand location. The first operand, a floating-point register, remains unchanged. The second operand must be eight bytes in length and requires fullword alignment.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Addressing
Protection (store violation)
Specification

STORE HALFWORD (STH)

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

 ___________________________________________________________
|               |  R   |  X   |  B   |                      |
|      40       |   1  |   2  |   2  |           2          |
|_______________|______|______|______|______________________|
 0               8      12     16     20                   31
```

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)
```

90	R 1	R 3	B 2	D 2
0	8	12	16	20 31

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

This manual is updated by: Addendum 800-1100PO-04.01

STORE SEGMENT CONTROL REGISTER (STSCTL)^{p}

STSCTL R1,R3,D2(X2,B2) (RS)

```
 ____________________________________________________________
|                 |  R   |  R   |  B   |                    |
|       A4        |   1  |   3  |   2  |         2          |
|                 |      |      |      |                    |
 ------------------------------------------------------------
0                 8      12     16     20                  31
```

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.

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. R1 and R3 must fall in the range 0-7, and 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

STORE MULTIPLE (STM)

```
STM     R1,R3,D2(B2)            (RS)
 _____________________________________________________________
|                | R    | R    | B    |            D          |
|       90       |   1  |   3  |   2  |             2         |
|________________|______|______|______|_______________________|
0                8      12     16     20                     31
```

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 SPECIAL REGISTER (STSREG)^P

STSREG D2(B1)   (S)

```
 ____________________________________________________________
|                |                |  B     |                |
|       9B       |       02       |  1     |       D        |
|                |                |        |       1        |
|________________|________________|________|________________|
0                8                16       20              31
```

Data is moved to memory from a 32-bit special register, which may be accessed only by the LSREG and STSREG instructions (and by STDD).

The contents of the special register are moved to the fullword at the address specified by the B1 and D1 fields. The address must be fullword aligned.

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Privileged operation
Access (store, operand 1)
Specification

SUBTRACT (SR, S)

```
SR   R1,R2                     (RR)

 ________________________________
|                |  R  |  R  |
|       1B       |  1  |  2  |
|________________|_____|_____|
0                8    12   15
```

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

 ___________________________________________________________
|                |  R  |  X  |  B  |                        |
|       5B       |  1  |  2  |  2  |           D2           |
|________________|_____|_____|_____|________________________|
0                8    12    16    20                       31
```

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.

SUBTRACT DECIMAL (FLOATING-POINT) (SQR, SQ)

```
SQR     R1,R2                  (RR)
 ___________________________________
|              |  R   |  R   |
|     3B       |   1  |   2  |
|______________|______|______|
 0              8     12     15

SQ     R1,D2(X2,B2)            (RX)
 ______________________________________________________
|              |  R   |  X   |  B   |           D      |
|     7B       |   1  |   2  |   2  |           2      |
|______________|______|______|______|__________________|
 0              8     12     16     20                31
```

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 DECIMAL (SP)

```
SP    D1(L1,B1),D2(L2,B2)        (SS)
______________________________________________  __________  _____
|               | L   | L   | B   | / /D |  B    |/ / D  |
|      FB       |  1  |  2  |  1  | - - 1|  2    |- -  2 |
|_______________|_____|_____|_____|/_/___|_______|/_/____|
0               8    12    16    20     32      36     47
```

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 HALFWORD (SH)

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

 ______________________________________________________________
|                 | R   | X   | B   |                          |
|       4B        |   1 |   2 |   2 |            2             |
|                 |     |     |     |                          |
 --------------------------------------------------------------
 0                 8    12    16    20                        31
```

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)

```
 ______________________________
|               | R   | R   |
|      1F       |   1 |   2 |
|_______________|_____|_____|
0               8    12    15
```

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

```
 ____________________________________________________________
|               | R   | X   | B   |                D         |
|      5F       |   1 |   2 |   2 |                2         |
|_______________|_____|_____|_____|__________________________|
0               8    12    16    20                         31
```

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)
______________________________
|             | | R  | R  |
|     2B      |0| 1  | 2  |
|_____________|_|____|____|
0              8,9  12  15

SER   R1,R2                   (RR, Short)
______________________________
|             | | R  | R  |
|     2B      |1| 1  | 2  |   (optional)
|_____________|_|____|____|
0              8,9  12  15

SD    R1,D2(X2,B2             (RX, Long))
______________________________________________________________
|             | | R  | X  | B  |          D              |
|     6B      |0| 1  | 2  | 2  |          2              |
|_____________|_|____|____|____|_________________________|
0              8,9  12   16   20                        31

SE    R1,D2(X2,B2             (RX, Short)
______________________________________________________________
|             | | R  | X  | B  |          D              |
|     6B      |1| 1  | 2  | 2  |          2              |   (optional)
|_____________|_|____|____|____|_________________________|
0              8,9  12   16   20                        31
```

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)

 ______________________________
|              |               |
|      0A      |       I       |
|______________|_______________|
 0              8             15
```

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 contents of bit positions 8 to 15 of the instruction are placed in the interruption code portion of this stored PCW. The contents of the three low-order bytes of control register 1 are pushed onto the stack, preceded by a byte containing X'01'. The contents of general registers 14 through 0, in descending order, are then pushed onto the stack. 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.

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

SUPERVISOR CALL EXIT (SVCX)^{p}

```
SVCX                        (RR)

 ______________________________
|              | R    |///////|
|      27      |   1  |///////|
|______________|______|///////|
0              8      12   15
```

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.

If the new active PCW has the single-step trap bit on, a single-step trap exception will occur immediately on completion of the instruction, even if the previously active PCW did not have the single-step trap bit on.

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)

```
 _____________________________________________________________
|               |       I       |   B    |          D          |
|      91       |       2       |   1    |          1          |
|_______________|_______________|________|_____________________|
0               8               16       20                  31
```

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)

TRANSLATE (TR)

```
TR   D1(L,B1),D2(B2)          (SS)

 ____________________________________________    _________    ______
|             |             |    B    | / /D |   B     |/ / D   |
|     DC      |      L      |    1    | --  1|   2     |--    2 |
|_____________|_____________|_________|_/_/__|_________|/_/_____|
0             8            16        20     32        36       47
```

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 entire initial address, the 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

If operand 2 can span a page (using the maximum table displacement, i.e., X'FF'), then operand 1 is scanned to determine the exact range of table locations to be referenced. The referenced virtual page or pages (two pages, at most) are checked to see if they reside in physical page frames. If an I/O operation should overlay operand 1 and invalidate this scan, then a page fault can be generated in the middle of the instruction, assuming the I/O operation caused a second (nonresident) virtual page to be referenced. This in turn will cause retranslation of part of operand 1 after the page fault has been serviced, and the instruction is re-executed.

TRANSLATE AND TEST (TRT)

```
TRT    D1(L,B1),D2(B2)          (SS)

 _________________________________________________________________
|               |               | B    | / /D |  B     |/ / D  |
|      DD       |       L       |  1   | - - 1|  2     |- -  2 |
|_______________|_______________|______|/ /___|________|/ /____|
0               8              16     20     32       36     47
```

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)

 ________________________________________________________________
|                  |  L  |  L  |  B   | / /D |  B   |/ / D  |
|        F3        |  1  |  2  |  1   | - - 1|  2   |- -  2 |
|__________________|_____|_____|______|_/_/__|______|/_/____|
0                  8     12    16     20     32     36     47
```

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)

```
 _______________________________________________   ___________   _______
|               |  L  |   L   |   B   | / /D |   B  |/ / D      |
|      F4       |  1  |     2 |    1  | - - 1|    2 |- -  2     |
|_______________|_____|_______|_______|/_/___|______|/_/________|
0               8    12      16      20     32     36          47
```

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)

 _________________________________________   _________   ____
|                  | L  | L   | B   | / /D | B   | / /D |
|       DB         |  1 |  2  |  1  |   1|  2  |   2  |
|__________________|____|_____|_____|/ /___|_____|/ /___|
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)

This manual is updated by: Addendum 800-1100PO-04.01

ZERO AND ADD (ZAP)

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

```
 ____________________________________________  __________  _____
|           | L  | L  | B  | / /D |  B  | / /D  |
|    F8     |  1 |  2 |  1 |     1|  2  |     2 |
|           |    |    |    |_/ /__|     |_/ /___|
0           8   12   16   20     32    36     47
```

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

7.2 OPERATING SYSTEM ASSIST INSTRUCTIONS

The following instructions, some of which are described in terms of equivalent pseudo-assembler-language instruction sequences, are intended for use by operating system routines only. They are reserved for use by software provided by Wang Laboratories, and their specification and function may change in the future. Use of these instructions other than by Wang Laboratories software development groups is discouraged.

All instructions in this section are privileged.

MODIFY TIMER QUEUE (MTQ)^{p}

MTQ   M1,D2(B2),D3(B3),R4   (SS)

```
 ______________________________________________   ______   ________
|               | R   | M   | B   | / /D  | B     | / /D  |
|      C7       | 4   | 1   | 2   | - - 2 | 3     | - - 3 |
|_______________|_____|_____|_____|_/_/___|_______|_/_/___|
0               8    12    16    20      32      36      47
```

This privileged instruction is used to set a clock comparator expiration value.

Register R4 contains a time interval value (in binary, in units of one clock tick). Operand 2 (D2(B2)) addresses a word-aligned, linked-list head, identical to the stack head word of the ENSK instruction. Operand 3 addresses a doubleword-aligned timer queue element. The first byte of this element is unchanged by the instruction. Bit 0 of this byte is tested; if it is 1, this element is to be removed from the list. The second through fourth bytes contain elements on the list in ascending order of expiration times. The fifth through eighth bytes contain the expiration time, in clock ticks, of the interval represented by this timer queue element.

The instruction operates as follows:

1. If bit 0 of the timer queue element (operand 3) is set, the list is searched for an element at the operand 3 address, and this element is removed from the queue. Queue elements examined must not be page faulted, lest the instruction give erroneous results.

   A specification exception will occur if any of the following are true: the addressed element is not on the queue, or removing this element empties the queue completely, or more than 256 elements are found on the queue before this element is encountered.

2. If bit 0 of the instruction's M1 field is set, the instruction is now terminated.

3. The time interval value in register R4 is added to the current clock value, and the result stored in the fifth through eighth bytes of operand 3.

4. The operand 3 timer queue element is placed on the queue in order of its expiration time (the value just placed in its fifth through eighth bytes). The second through fourth bytes of operand 3 are used as the list chain word.

   A specification exception occurs if the chain is empty, or if more than 256 elements are encountered before the operand 3 timer queue element is inserted.

5. The contents of the fifth through eighth bytes of the first element of the timer list are placed in the clock comparator (control register 7).

Resulting Condition Code

The condition code remains unchanged.

Program Exceptions

Access (fetch and store, operands 1 and 2)
Specification
Privileged Operation

Programming Note

The specification exception taken if more than 256 elements are on the timer queue is intended to prevent the instruction from 'hanging up' the CP if the list elements are chained in a loop.

SCAN PAGE FRAME TABLE (SPFT)^{p}

```
SPFT    R1,R3,D2(B2)            (RS)

 ____________________________________________________________
|               | R  | R  | B  |                              |
|      AE       | 1  | 3  | 2  |              2               |
|               |    |    |    |                              |
 ------------------------------------------------------------
 0              8    12   16   20                            31
```

R1 format:

```
 ____________________________________________________________
|               |              |              |               |
|               |    LRURG     |     SCC      |      RGL      |
|               |              |              |               |
 ------------------------------------------------------------
 0              8              16             24              31
```

LRURG - Least recently used reference group
SCC - Scan class comparand
RGL - Reference group limit

R3 format:

```
 ____________________________________________________________
|               |                                             |
|     PFTL      |                   SDISP                     |
|               |                                             |
 ------------------------------------------------------------
 0              8                                             31
```

PFTL - Page frame table length
SDISP - Starting PFT displacement

Page Frame Table format:

```
 _____________________________________________  ______________
|               |              |C|     / /                    |
|      RGN      |      SC      |B|   for software use         |
|               |              | |___/ /______________________|
 ------------------------------------------------------------
 0              8             16 17                           63
```

RGN - Reference group number
SC - Scan class
CB - 'Change' bit

The page frame table physical address (operand address D2(B2)), which must be doubleword aligned, is added to the SDISP field of the register addressed by R3. If the page frame table address is not doubleword aligned or if the contents of the SDISP field are not a multiple of 8, a specification exception occurs and the instruction is suppressed. Each page frame table entry beginning at the resulting address is examined, where the total number of entries assumed to be in the table is given by the PFTL field of the register addressed by R3. The SC field of each entry is compared with the SCC field of the register addressed by Rl. If the SCC field of this register is nonzero and the two fields compared are unequal, the SDISP field of register R3 is increased by 8 and the next page frame table entry is processed. If the SCC field is 0 or the two fields compared contain equal values, the reference bit for this page frame is examined in the local page frame table. If it is 0 it is set to 1, binary 0s are placed in field RGN of this page frame table entry, and the change bit for this page frame is inclusive ORed with bit CB of the page frame table entry. The SDISP field of register R3 is then increased by 8 and the next page frame table entry is processed.

If the reference bit for the page frame is 1 when examined, then field RGN of the page frame table entry is compared with field RGL of register Rl. If they contain equal values, then the SDISP field of register R3 is increased by 8 and the next page frame table entry is processed. If they contain unequal values, the following processing occurs:

- If field RGN plus 1 is equal to field RGL, then field RGN is incremented by 1 and the instruction completes with condition code 2 set.

- Otherwise, if field RGN (unincremented) is greater than or equal to field LRURG of register Rl, then field RGN is incremented by 1 and the resulting value in field RGN is placed in field LRURG. The instruction is then completed with condition code 1 set.

- Otherwise (field RGN is not equal to field RGL minus 1 and is not greater than field LRURG), field RGN is incremented by 1, the SDISP field of register R3 is increased by 8 and the next page frame table entry is processed.

If the end of the page frame table (last entry is at address D2(B2) plus 8 times PFTL, minus 8) is reached without another completion for the instruction (including the cases of PFTL being 0 and of PFTL times 8 being less than or equal to SDISP when the instruction is initiated), then the instruction is completed with condition code 0 set.

Resulting Condition Code

0 End of table reached.

1 RGN-1 greater than LRURG for table entry at second operand address plus SDISP.

2 RGN equal to RGL for table entry at second operand address plus SDISP.

Program Exceptions

Privileged operation
Access (fetch and store, operand 2)
Specification

CHAPTER 8
INPUT/OUTPUT OPERATION

8.1 INTRODUCTION

Transfer of information between main memory and input/output (I/O) devices is referred to as an I/O operation. These operations use I/O processors (IOPs), which contain the logic circuitry controlling the transfer of data between devices and main memory. An I/O Command Word (IOCW), containing a command to be performed, is sent from the CP to the device when the I/O operation is initiated. An I/O Status Word (IOSW), reporting on the execution of this command, is sent back later from the device to the CP.

Chapters 8 through 12 of the manual describe the programmed control of I/O devices by the system. Formats are defined for the various types of I/O control information. The formats apply to all I/O operations and are independent of the types of I/O devices, their speed, or mode of operation.

The formats described include provision for functions unique to particular devices. The way in which a device makes use of each format is defined in the chapter for that device.

8.2 I/O SUBSYSTEM

8.2.1 IOPs

At the beginning of an I/O operation, the CP issues a privileged START I/O (SIO) instruction to the IOP (for the VS25, to the BP) controlling the I/O device that is to perform the operation. If both the IOP and the indicated device can service the request they begin to do so, and there is no further communication between the CP and IOP until the IOP issues an I/O completion interrupt request. Upon receiving the SIO, the IOP accesses the indicated IOCW for further information (e.g., the command, the type of operation (READ, WRITE, etc.), and the location of the data to be transferred. The IOP interprets the command for the device, routes the data to or from main memory, and performs I/O error checking and correction. Finally, the IOP makes an I/O completion interrupt request to the CP and stores information regarding the completed operation in the IOSW.

8.2.2 I/O Devices

I/O devices enable communication between data processing systems and their environment, and also provide for storage of information outside main memory. Such devices include workstations (terminals), magnetic tape drives, disk drives, printers, and teleprocessing equipment.

NOTE

The I/O subsystem operates independently of virtual memory; therefore, all addresses relating to I/O functions must be physical addresses.

8.2.3 VS25 and VS100 Device Address (IODA)

VS25 and VS100 systems use the low-order 16 bits of the general register specified in an SIO, CIO, or HIO instruction to give the address (IODA) of the I/O device involved. The interpretation of these bits is as follows:

```
         _____________________________
        |    |    |                   |
        |BA# |IOP |      Port #       |
        |____|_#__|___________________|
bits     0    3    6                15
```

Figure 8-1. I/O Device Address (IODA)

Bits 0-2 specify the BP (for the VS25) or BA (for the VS100). For the VS25, designations other than '001' cause a specification exception; for the VS100, designations other than '001' (BA1) or '010' (BA2) cause a specification exception.

Bits 3-5 for the VS25 specify one of 3 DAs: '000' for the diskette DA, '001' for the fixed disk DA, or '010' for the serial devices DA. For the VS100, 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 port number on the VS25 DA or VS100 IOP. The port number is an index into the device configuration table (DCT) for DAs and IOPs using a DCT. Current DAs and IOPs do not support the potential 10-bit range of port numbers, and reject instructions containing invalid port numbers.

8.2.4 VS80 I/O Device Identification

Each VS80 device has a 1-byte device address, for which all values from X'00' to X'FE' are legitimate. (Note that the priority of interrupt service for the VS80 is determined by the physical position of the device's IOP in the hardware configuration and not by the device address.) A VS80 device address consists of an IOP portion and a device portion; because current IOPs support either 4, 16, or 32 devices apiece, the high-order 6, 4, or 3 bits are the IOP portion, and the low-order 2, 4, or 5 bits are the device portion.

8.3 MEMORY ASSIGNMENTS FOR INTER-PROCESSOR COMMUNICATIONS

8.3.1 VS25 and VS100 Assignments

Table 8-1 summarizes permanent memory assignments for the VS25 and VS100:

Table 8-1. VS25, VS100 Permanent Memory Assignments

Location	Data Item	Written By	System
X'00-07'	IOSW	CP	VS25, VS100
X'50'	SQB	CP	VS25, VS100
X'80-81'	IODA (on I/O interrupt)	CP	VS25, VS100
X'82-8F'	(for VS100, area reserved for system use; for VS25, CP-BP communications area)		
X'90' (start of DAST)	IOSW, SQB	BP	VS25
X'90' (start of IOPST)	IOSW	IOP	VS100
per CTA (start of IOCT)	IOCW	CP	VS25, VS100

Communications Areas Common to VS25 and VS100

Physical Addresses X'00-07': here the CP microprogram writes the IOSW associated with the I/O interrupt just granted.

Physical Address X'50': here the CP microprogram writes the SQB associated with the I/O interrupt just granted. (Refer to Subsection 8.3.5, below, for details on the SQB.)

Physical Address X'80': here the CP microprogram writes the IODA associated with the I/O interrupt just granted.

VS25-Specific Communications Area

In addresses X'82-8F' the CP writes (for the BP) IOCWs, IODAs, and other information associated with upcoming I/O commands, and the BP writes IODAs associated with upcoming I/O completion interrupt requests.

8.3.2 VS25 DA Status Table (DAST)

The VS25's DA Status Table (DAST) begins at location X'90', and consists of one 16-byte entry for each DA attached to the system. The entries are arranged in ascending order by DA number. The format of each entry is as follows:

```
        ________________________________
        |             |S|        |   |
        |     IOSW    |Q| Res.   |CTA|
        |_____________|B|________|___|
Bytes   0              8 9      13  15
```

Figure 8-2. DA Status Table (DAST) for the VS25

The leftmost half of each DAST entry consists of an IOSW. It is in this location that the BP pre-stores the IOSW upon completion (successful or otherwise) of an I/O operation, i.e., when an I/O completion interrupt is requested. Immediately following the IOSW it pre-stores the status qualifier byte (SQB--see also Subsection 8.3.5, below). When the I/O interrupt is granted, the CP microprogram copies the IOSW to location X'00', and the SQB to location X'050'.

Immediately following the SQB is a four-byte area reserved for VS25 use.

The rightmost 3 bytes of each DAST entry are the command table address (CTA) of the I/O Command Table (IOCT) for the particular DA. Refer to Subsection 8.3.4, below, for information about the IOCT.

8.3.3 VS100 IOP Status Table (IOPST)

The VS100's IOP Status Table (IOPST) begins at location X'90' and consists of one 16-byte entry for each IOP attached to the system. The entries are arranged from IOP0-IOP7 on BA1, then from IOP0-IOP7 on BA2. The format of each entry is as follows:

```
        __________________________________
        |             |          |   |
        |     IOSW    | Res'd.   |CTA|
        |_____________|__________|___|
Bytes   0              8        13  15
```

Figure 8-3. IOP Status Table (IOPST) for the VS100

The leftmost half of each IOPST entry consists of an IOSW. It is to this location that an IOP writes the IOSW upon completion (successful or otherwise) of an I/O operation, i.e., when an I/O completion interrupt is requested. When the I/O interrupt is granted, the CP microprogram copies the IOSW from its location in the IOPST to location X'00'.

The rightmost 3 bytes of each IOPST entry are the command table address (CTA) of the I/O Command Table (IOCT) for the IOP.

8.3.4 VS25 and VS100 I/O Command Table (IOCT)

The IOCT for each VS25 DA or VS100 IOP may begin at any doubleword-aligned address in main memory, and consists of one 16-byte entry for each device attached to the DA or IOP. The entries are arranged from port 0 to port n, where port n is the highest-numbered port through which a device is attached to the particular DA or IOP.

Each IOCT entry begins with a 9-byte IOCW for the particular device. The remaining 7 bytes are reserved for use by the outer program. The VS100 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.

The format of each IOCT entry is as follows:

```
         _____________________________
        |               |      |     |
        |     IOCW      | Res. |UCBA |
        |_______________|______|_____|
Bytes   0               9     13    15
```

Figure 8-4. I/O Command Table (IOCT)

8.3.5 VS25 and VS100 Status Qualifier Byte (SQB)

During certain critical operations required to control its I/O devices, an IOP (for the VS25, the BP) may be unable to accept a CIO, SIO, or HIO. This condition is of limited duration; its frequency depends on the device. VS25 and VS100 systems alert the CP to such a condition by writing a one-byte extension of the upcoming IOSW for the IOP. At location X'50' (the interrupt code byte of the Old I/O Interrupt PCW), this Status Qualifier Byte (SQB) is written in the following format:

```
         _______________
        |  |       |R|R|
        |00|Res'd. |I|D|
        |__|_______|P|B|
bits     0  2      5 6 7
```

Figure 8-5. Status Qualifier Byte (SQB)

The SQB is used by both the VS25 and VS100 to further describe an I/O command rejected by a particular device:

the RIP bit is set to indicate "Rejected--interrupt pending" if the new CIO or SIO command is rejected because the particular device has an unsolicited interrupt to report but the IOP (for the VS25, the BP) has not yet raised its request line, i.e., has not yet stored the IOSW in its IOPST (for the VS25, its DAST). An HIO instruction is ignored in this case.

the RDB bit is set to indicate "Rejected--device busy" if the new CIO or SIO command is rejected because the particular device is executing a previous command and so has not yet requested a completion interrupt. The RDB bit is not set for a new HIO instruction.

for the VS25 only: both bits are set (i.e., SQB=3) to indicate that the DA is not physically present.

8.3.6 VS80 Communications Areas

VS80 IOCA

The VS80 I/O Command Area (IOCA) area starts at main storage location 128 and contains a halfword entry for every value from 0 to 255, which is the highest possible device address. The IOP uses the device address received on an SIO or CIO instruction as an index into the IOCA area. Each IOCA entry contains a halfword physical address for the IOCW to be executed; thus, these addresses cannot be greater than X'FFFF', and must specify locations within the first 64K bytes of main memory.

VS80 Handling of "IOP Busy" and "Device Busy"

When a VS80 IOP is unable to accept an SIO or HIO, a condition code indicating "IOP Busy" is returned. The circumstances causing this response are device dependent. Once an IOP has responded to an instruction with an IOP Busy interrupt, it will present an "IOP Now Ready" interrupt after the busy condition clears. Only one IOP Now Ready will be presented no matter how many SIOs, CIOs, or HIOs are rejected. The IOP Now Ready may be presented to the CP with the next IOSW for any of the devices attached to that IOP, or as a separate interrupt with the device portion of the indicated device address set to zero (whether or not this device is attached to the system).

If the device either is busy with a previous I/O command or has a pending request for an I/O completion or unsolicited interrupt, the SIO will be terminated with a "Device Busy" indication.

If the device is not attached or is unable to complete the I/O operation, the SIO is accepted and "Device Not Ready--Intervention Required" is reported on an I/O interrupt.

8.3.7 Resetting of I/O Devices--All Systems

All I/O devices are reset when the LOAD button is pushed or when a system power-on sequence is completed. Resetting causes I/O devices to terminate all I/O 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.

8.4 EXECUTION OF I/O OPERATIONS

I/O devices can execute three commands: WRITE, READ, and CONTROL (no data transfer); each command initiates a corresponding I/O operation or activity in the device. The next I/O command to be executed by each device is contained in the IOCW for the device, which in turn is written in the appropriate entry of the IOCT. Refer to Subsection 8.3.4 for a discussion of the IOCT. (For the VS80, the address of the IOCW is found in the IOCA, described in Subsection 8.3.6.)

8.4.1 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 IOP (for the VS25, via the BP and a DA). The CIO instruction starts control operations for the IOP, 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:

```
      _______________________________
      |                |     |/////|
      |     Opcode     | R1  |/////|
      |________________|_____|/////|
bits   0                8    12   15
```

Figure 8-6. 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 the device address of the I/O device involved in the operation, in the following format:

```
      ____________________________________________
      |//////////////////|    |    |             |
      |//////////////////|BA#|IOP|   Port #    |
      |//////////////////|____|____|_____________|
bits   0                 16   19   22           31
```

Figure 8-7. R1 Format for SIO, CIO, and HIO Instructions

Note that this address is simply the IODA, written into the rightmost half of the register. This address is used directly by the IOP as an index into its IOCT, where the IOCW for the device is written, giving further information about the I/O operation (e.g., data address and data length).

8.4.2 Transmission of SIO

The SIO instruction is sent from the CP to devices through an intermediate processor.

VS25 Receipt of SIO

For the VS25, the CP sends the SIO instruction not directly to the DA but to the BP, and sets a condition code according to the result as follows:

Condition Code	Meaning
0	Command received by the BP
1	not used
2	not used
3	BP busy

A setting of 0 means typically that the device has accepted the I/O command; in some cases where a setting of 0 is returned, the BP may still reject the command, setting the RIP or RDB bit in the SQB extension of the upcoming IOSW. A setting of 3 means that a previous I/O instruction has not yet been read by the BP.

VS100 Receipt of SIO

For the VS100, the CP sends the SIO instruction not directly to the IOP but to the BA for that IOP, and sets a condition code according to the result as follows:

Condition Code	Meaning
0	Command received by the IOP
1	not used
2	IOP busy
3	IPC-IN register busy

A setting of 0 means typically that the device has accepted the I/O command; in some cases where a setting of 0 is returned, the IOP may still reject the command, setting the RIP or RDB bit in the SQB extension of the upcoming IOSW. A setting of 2 means that an I/O operation is already in progress for some device on the IOP. This condition has ended when a subsequent IOSW from the IOP, with its IOP Now Ready bit set, is processed by the CP. A setting of 3 means that a previous I/O instruction has not yet been read by the IOP.

VS80 Receipt of SIO

The VS80 CP sends an SIO instruction to the IOP for the involved device; the IOP uses the included device address as an index into the IOCA area, to find the address of the associated IOCW.

8.4.3 I/O Command Word (IOCW) for SIO Instruction

The IOCW specifies the command to be executed. For 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. (For the VS80, the location of the IOCW is specified by the IOCA at SIO time.)

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 IOP will change the IOCW or IOCA.

The IOCW consists of a 6-byte general section and a variable-length device-dependent section, as shown in Figure 8-8. The device-dependent section can be of any length, but is fixed for each device. The IOCW must be fullword aligned. Examples of the IOCW are given in Table 8-2.

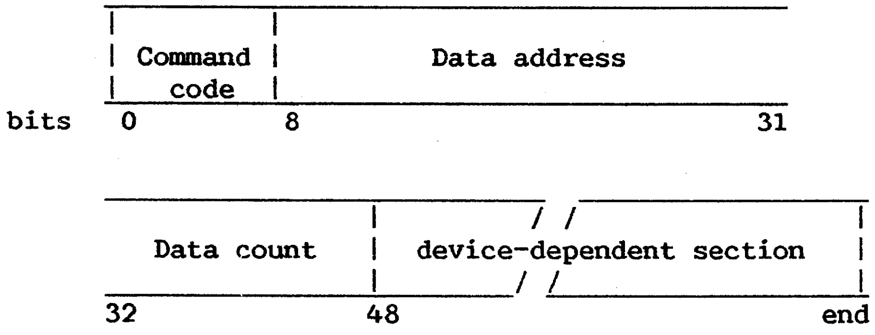


Figure 8-8. I/O Command Word (IOCW) Format

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.

```
 ________________________
|                        |
|      IOCW address      |
|________________________|
0                      15
```

Figure 8-1. IOCA Format

The IOCA is a 16-bit address of the IOCW. All IOCWs must start within the first 64K bytes of physical memory.

8.4.2 I/O Command Word (IOCW) for SIO Instruction

The IOCW specifies the command to be executed. For commands initiating data transfer, it designates the storage area associated with the operation. The location of the IOCW is specified by the IOCA at SIO time. From the time an SIO is accepted until the clearing of the I/O completion interrupt, the IOCW must not be changed. The device and IOP will not change the IOCW or IOCA.

The IOCW consists of a 6-byte general section and a variable-length device-dependent section, as shown in Figure 8-2. The device-dependent section can be of any length, but is fixed for each device. The IOCW must be fullword aligned. Examples of the IOCW are given in Table 8-1.

```
 ______________________________________________________
|             |                                        |
| Command code|              Data address              |
|_____________|________________________________________|
0             8                                      31
 ______________________________________________________
|                   |                                  |
|    Data count     |    device-dependent section      |
|___________________|__________________________________|
32                47 48                              end
```

Figure 8-2. I/O Command Word Format

The fields in the IOCW are allocated for the following purposes:

- Command code--Bits 0-7 specify the operation to be performed.

- Data address (DA)--Bits 8-31 specify the physical address of an 8-bit byte in main memory, which must be fullword aligned. This byte location is the beginning of the data area for the specified operation, or is the beginning of an Indirect Data Address list, which in turn addresses the data area(s) 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.

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'.

"Reserved" means reserved for system use.

A READ operation is initiated at the I/O device, and data is transferred from the device to main memory. Data in memory is placed in ascending order of addresses, starting with the address specified in the IOCW.

A WRITE operation is initiated at the I/O device, and data is transferred from main memory to the I/O device. Data in memory is fetched in ascending order of addresses, starting with the address specified in the IOCW.

A CONTROL operation is initiated at the I/O device. A CONTROL command is used to initiate an operation not involving transfer of data. 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.

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 which support that option.

When the IOCW designated contains an invalid field, an I/O interrupt is generated with the invalid condition indicated in the IOSW.

Programming Note

The IOCW must not be changed between the SIO and the I/O interruption.

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 8-bit byte to be transferred and the number of consecutive 8-bit 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).

In the event 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 means of 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, on 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.

Indirect Address Lists

Certain devices expedite the transfer of more than one page 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 (2K-byte) 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.

Certain devices (especially disk devices) may require that the memory addresses specified in Indirect Address list entries have up to 11 low-order 0s (i.e., be aligned on a boundary as large as 2K bytes). Refer to specific device descriptions for the restrictions applicable to particular devices.

Device-Dependent Section

This section of the IOCW is not required by all devices. The length is specified with the device description. One use for this area is the sector address for a disk drive.

8.5 TERMINATION OF I/O OPERATIONS

The following sequence of events occurs when I/O operations are performed.

1. Pending interruption is established.

2. Interruption remains pending until it is either accepted by the CP as an I/O interruption or cleared by an HIO.

3. IOSW is stored in fixed low storage when the pending interrupt is accepted. (The IOSW is explained in Section 8.6.)

I/O completion is caused either by the operation's being normally completed, with or without errors, or as a result of an HIO. When the pending interrupt is cleared, the I/O completion status is available in the IOSW.

An IOSW will not be stored a second time. It will be available until overlaid when another pending interrupt is cleared. To guarantee the validity of an IOSW, it should be moved from the IOSW area in code disabled for I/O interruption. This code must have been disabled by the loading of the I/O New PCW or must have been disabled when the SIO was issued.

8.5.1 Types of Termination

Normally an I/O operation lasts until the device completes the operation. When a system load or power-on is performed, all I/O operations are terminated immediately. The system can force an I/O operation to terminate prematurely by issuing an HIO.

Termination of Data Transfer

When the device accepts a data transfer command, the operation will be terminated by one of the following five conditions:

1. A HALT I/O instruction was issued to the device.

2. The count field in the IOCW has gone to 0 (IOCW exhaustion).

3. As many bytes have been transferred as are indicated by the sum of the lengths specified in an Indirect Address List (list exhaustion).

4. The device has indicated that there is no more data to be transferred (data exhaustion).

5. Hardware malfunction.

The end condition causes the operation to be terminated and an interruption condition to be generated. The status bits in the associated IOSW indicate the reasons for termination. The device can signal termination at any time after initiation of the operation, and the signal may occur before any data has been transferred. The duration of data transfer operations is variable and is controlled by the device and its IOP.

Termination by HALT I/O

If accepted by the IOP, the instruction HALT I/O causes the current operation at the addressed device to be terminated immediately. If an interruption for the addressed device was pending, that interruption remains pending. If an I/O operation was active, the operation is terminated and a completion interruption becomes pending.

Termination Due to Equipment 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 soft error. If the recovery is unsuccessful, the operation is terminated and a hard error is indicated in the IOSW.

8.5.2 I/O Interruptions

I/O interruptions provide a means for the system to change its state in response to conditions that occur in I/O devices and IOPs. These conditions are caused by termination of an I/O operation or by operator intervention at the I/O device.

These conditions cause three types of I/O interruptions: solicited, unsolicited, and IOP NOW READY. A solicited interruption is caused by the completion of an I/O operation initiated by an accepted SIO or CIO. An unsolicited interruption is caused by operator action at the I/O device such as mounting a disk pack or striking a workstation attention key. An IOP NOW READY interruption is caused by an IOP's becoming available for acceptance of SIOs, CIOs, and HIOs after having reported IOP BUSY in response to one of these instructions.

When multiple I/O interruption requests are pending, the hardware establishes a priority sequence for them before initiating an I/O interruption request. While the processor is servicing one interrupt, the others remain pending.

Interruption Conditions

The conditions causing requests for I/O interruptions to be initiated are called I/O interruption conditions. An I/O interruption condition can be brought to the attention of the system program only once and is cleared when it causes an interruption. The device or IOP attempts to initiate a request to the processor for an interruption when any of the following conditions (not all of which are defined for every device and IOP) occur:

1. Attention or device now ready
2. IOP now ready
3. I/O completion
4. Intervention required.

8.5.3 Priority of Interrupts

All I/O interrupt requests 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 IOP in the hardware configuration, which is determined at installation time.

8.5.4 Interrupt Action

I/O interrupts are handled as described below for the various VS CPs.

VS25 Interrupt Processing

BPs report the completion of any I/O operation (whether successful or not) at one of their devices by a solicited I/O interrupt. At the time of a request for an I/O interrupt, the BP has already pre-stored the IOSW, SQB, and IODA in the appropriate locations (refer to Section 8.3, above) in main memory.

The CP grants an interrupt after ascertaining the DA and device number of its origin from the pre-stored IODA. It then copies the IOSW into location X'00' of main memory, the SQB into location X'50', and the IODA 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.

VS100 Interrupt Processing

IOPs report the completion of any I/O operation (whether successful or not) at one of their devices by a solicited I/O interrupt. 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 Subsection 8.3.3, above) and has pre-stored the port number of the device, 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 IODA (formed by combining the BA number, IOP number, and port number of the involved device) 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.

8.6 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 summarized in Figure 8-9:

```
        _____________________________________________________________________
        |        |        |                |               |   device-      |
        | General| Error  |    device-     |   Residual    |   dependent    |
        | status | status |   dependent    |  byte count   |  (extended)    |
   bits  0        8        16               32              48              63
```

Figure 8-9. IOSW Format

Examples of the IOSW are given in Table 8-2.

Table 8-2. IOCWs and IOSWs (from the I/O Error and IPL Log)

I/O Command	Error Description	IOCW	IOSW
READ	soft error	43 000F44 0800 000000	60101008 00000000
WRITE	hard error	82 00F75C 0054 000047	20103F00 00000000

Subsections 8.6.1 through 8.6.4 provide an overview of the fields in the IOSW, which are discussed in more detail in Sections 8.7 and 8.8. One byte of the IOSW, the general status byte, will always be stored. Additional bytes are stored as required by particular devices. A given type of device always stores an IOSW of the same length.

8.6.1 General Status Byte

The general status byte 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
5-6		Reserved--always 0
7		Reserved for software use

8.6.2 Error Status Byte

This byte 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.

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.

8.6.3 Device-Dependent Status Bytes

These two bytes (bits 16-31) are different for each type of I/O device. The use of these bytes is described along with device interfaces in Chapters 9-12 of this manual.

Subsections 8.6.1 through 8.6.4 provide an overview of the fields in the IOSW, which are discussed in more detail in Sections 8.7 and 8.8. One byte of the IOSW, the general status byte, will always be stored. Additional bytes are stored as required by particular devices. A given type of device always stores an IOSW of the same length.

8.6.1 General Status Byte

The general status byte 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
5-6		Reserved--always 0
7		Reserved for software use

8.6.2 Error Status Byte

This byte 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.

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 I/O processor before any normal I/O operation can be performed on programmable devices.
	=10 (PP)	A peripheral processor microprogram is required by the I/O processor before any normal I/O operation can be performed on programmable devices.
	=01 (DP)	A device processor microprogram is required by the I/O processor before any normal I/O operation can be performed on programmable devices.

8.6.3 Device-Dependent Status Bytes

These two bytes (bits 16-31) are different for different devices. The exact description of the use of these bytes can be found under the specific device description in Chapters 9-12 of this manual.

8.6.4 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 the data count. If the device does support it, this field will always be stored if IL is set. If the device does not support storing of the residual byte count, it may still set the IL bit.

Bits	Meaning
32-47	Byte count

8.7 GENERAL STATUS BYTE

8.7.1 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, or that no device with the specified device number was attached to the specified I/O processor. 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.

8.7.2 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.

8.7.3 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 IOP. If the EC bit is set, the errors detected will be indicated in the error status byte or device-dependent status bytes, whether or not NC is also set. Possible combinations 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

8.7.4 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. This indicates that either the device has become available for I/O operations or that someone is signaling the CP for attention. This bit is independent of, but may be set with, the NC, EC, and/or PC bits on.

8.7.5 PC--IOP Now Ready

This is an indication that an IOP may now accept an SIO. This bit can be set in conjunction with NC or EC (I/O completion) or U (unsolicited). Whenever an SIO is rejected with condition code 2 (IOP BUSY), an interruption with PC set will eventually be presented. If more than one SIO to devices on the same IOP is rejected with condition code 2 without an intervening interruption with PC set, then only one interruption with PC set will be presented.

8.8 ERROR STATUS BYTE

8.8.1 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.

8.8.2 MPE--Memory Parity Error

A memory parity error is indicated whenever there is a parity error while the IOP associated with the I/O device is accessing memory. This is the method by which a machine check is indicated during an I/O operation.

8.8.3 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. This is the method by which an addressing exception is indicated during an I/O operation.

8.8.4 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.

8.8.5 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.

8.8.6 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 error completion bit (EC) will be 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.

8.8.7 PP and DP--IOP or Device Code Not Loaded

For programmable IOPs (22V06, 22V07, and 22V17) and programmable devices (all models whose model numbers include the letter "S," "C," or "R"), 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 22V06 IOP in order to process an I/O operation.
	1	0	Microprogram reloading is required for the peripheral processors of the 22V06, 22V07, and 22V17 IOPs.
	0	1	Microprogram reloading for the programmable device (e.g., 2246S, 2221V-S) is required in order to process an I/O operation.

8.9 THE CIO INSTRUCTION

The CIO instruction directs I/O processors to perform diagnostic, microcode-loading, microcode-reading, and processor control functions. Completion interruptions are presented as for SIO-initiated operations.

8.9.1 CIO Microcode-Loading, Microcode-Reading, and Processor Control Commands

Figure 8-4 shows the format for microcode commands.

Command code	Memory address	Data count
0	1	4 5

Figure 8-4. IOCW Format for Microcode Commands

Byte 0--Command code: 10xxxxxx (WRITE), 01xxxxxx (READ), or 11xxxxxx (CONTROL), where xxxxxx is interpreted by the I/O processor to designate specific functions. These commands are not accepted by all I/O processors.

The following CIO commands are supported by the 22V06-1, 22V06-2, and 22V06-3 programmable telecommunication I/O processors:

Command	Meaning
1000 0000	Load device control table
1000 0010	Load device control table, with Indirect Data Addressing (IDA)
1001 0000	Load data link processor
1001 0010	Load data link processor, with IDA
1101 0000	Start data link processor
1100 0000	Restart bus control processor
0101 0000	Read data link processor's memory
0101 0010	Read data link processor's memory, with IDA
0100 0000	Read device control table
0100 0010	Read device control table, with IDA passed to data link processor
1010 1001	Activate or deactivate a remote device
1010 1011	Activate or deactivate a remote device, with IDA
All other	By convention, 1010 0000 is used to specify loading of device processor microcode, and 1110 0000 is used to specify starting of a device processor.

(For commands 1010 1001 and 1010 1011, a 4-byte data count field of the following form is used:

Cluster	Port	Device	Type
4	5	6	7

If Cluster = X'FF', then "deactivate" is indicated.)

The following CIO commands are supported by 22V07-1 and 22V07-2 serial-device IOPs:

Command	Meaning
1010 0000	Load device processor microcode
1110 0000	Start device processor

8.9.2 CIO Memory Diagnostic Commands for IOPs

Figure 8-11 illustrates the format of the IOCW for diagnostic commands.

	Command code	Memory address (M)	Data count (R)	Pattern	Index (I)
Bytes	0	1	4	6	8

Figure 8-11. IOCW Format for Diagnostic Commands

Byte 0 - Command code: 00 cca000, where

cc = 10 - Reference memory (REF)
cc = 01 - Modify memory (MOD)
a = 0 - Access 2 bytes
a = 1 - Access 1 byte

REF means that the specified memory area will be read as determined by the increment or index field, and this data will be compared with the given data pattern in the IOCW. An error is detected if the comparison is unequal.

MOD means that the specified memory area will be modified with the given data pattern in the IOCW. Immediately after each memory location is modified (a 1- or 2-byte write to memory), it will be read and compared with the given pattern. An error is detected if the comparison is unequal.

Bytes 1-3 - Memory address field (M): specifies the starting address of the memory area to be accessed.

Bytes 4-5 - Data count field (R): specifies the number of memory accesses (of 1 or 2 bytes) to be performed. Note that this is not a byte count.

Bytes 6-7 - Data pattern field: specifies the data value to be used for comparison.

Byte 8 - Index field (I): the increment for the memory address update (e.g., I = 0 indicates same main memory location to be accessed R times, and I = 4 indicates 1 or 2 bytes to be accessed for every word, starting from location M and finishing at location M+I*(R-1) if physical memory size permits).

NOTE

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

The IOP that receives such a CIO command will initiate the specified memory access starting from the given memory address and continuing until the specified data count is exhausted. A normal completion will then be reported in the IOSW. In case of any data error, the IOP will terminate immediately and report an error condition. The IOSW format for CIO commands is illustrated in Figure 8-12.

	Status	Error	Data obtained	Residual count
Bytes	0	1	2	4

Figure 8-12. IOSW Format for CIO Commands

IOSW byte 0 and byte 1 are the usual general and error status bytes.

Byte 0 - General status byte.

Byte 1 - Error status byte.

Bytes 2-3 - Contain the data value last obtained. For normal completion, this value should be equal to.the data pattern specified in the IOCW. For error completion, the bits in error are those in this field that differ from their corresponding bits in the given pattern. Note that for a 1-byte memory access, only the first byte is used.

Bytes 4-5 - Contain the residual data count (in number of memory accesses of 1 or 2 bytes each). For normal completion, the count is 0. For error completion, the residual data count can be used to compute the offset from the starting memory address of the memory location at which the error is detected.

CHAPTER 9
WANG WORKSTATION CHARACTERISTICS

9.1 INTRODUCTION

The VS workstation is designed both to simplify the operator's job and to reduce the processing time required by the central processor to handle its I/O. This device has two main parts, the CRT and the keyboard.

9.2 THE CRT

9.2.1 Screen and Cursor

The CRT screen is capable of displaying 24 rows of 80 characters each. Every position of the screen is capable of displaying any character. A special symbol (resembling an underscore) 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 keyed 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 9-1 gives the set of displayable characters and the associated representations in bits.

Table 9-1. 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
				High-Order Digit →	0	1	2	3	4	5	6	7
b_4 ↓	b_5 ↓	b_6 ↓	b_7 ↓	Low-Order Digit ↓								
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.

9.2.2 Workstation Memory

The internal memory of the workstation uses eight bits per character.

9.2.3 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 preceding the field. The possible attributes are defined in Table 9-2.

Table 9-2. 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 keyable 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 non-displayed 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. (See the description of the field attribute character, below.) 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.

9.2.4 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 to be written by WRITE SELECTED.

Underscore: The characters in this field are to be underscored when displayed on the screen.

Intensified display: The characters in this field will be displayed in higher intensity than those in a low-intensity display field.

Low intensity display: The characters in this field will be displayed on the screen.

Blinking display: The characters in this field will be displayed alternately in intensified display and display mode. The display will change modes at a fixed rate of about three times a second.

Nondisplay: The characters in this field will not be 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 keying of any character on the keyboard.

Uppercase shift: Letters will be displayed and stored as uppercase only. This is without regard to whether the shift or lock key is pressed. All other keys will 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.

9.2.5 Tabs

Ten tabs can be set by programs; they can be set to any column of the workstation's screen (1-80) with the instruction 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. Tabs have no effect within protected fields or during back-tab operations. When the workstation is powered on, all tabs are cleared.

9.2.6 Audio Indicators

The audible alarm sounds a short tone whenever an illegal keying operation is attempted. This operation can be the user's attempting to enter data into a protected field, trying to move the cursor past the end of the screen with a field-sensitive key, or trying to enter data when the keyboard is locked. The alarm is also sounded when a WRITE is issued while the proper bit is on in the WCC.

The keystroke indicator is a small device attached to the keyboard that makes a clicking sound. It sounds each time a key is pressed.

9.3 THE KEYBOARD

The VS workstation keyboard is illustrated in Figure 9-1.

9.3.1 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 of 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 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 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.

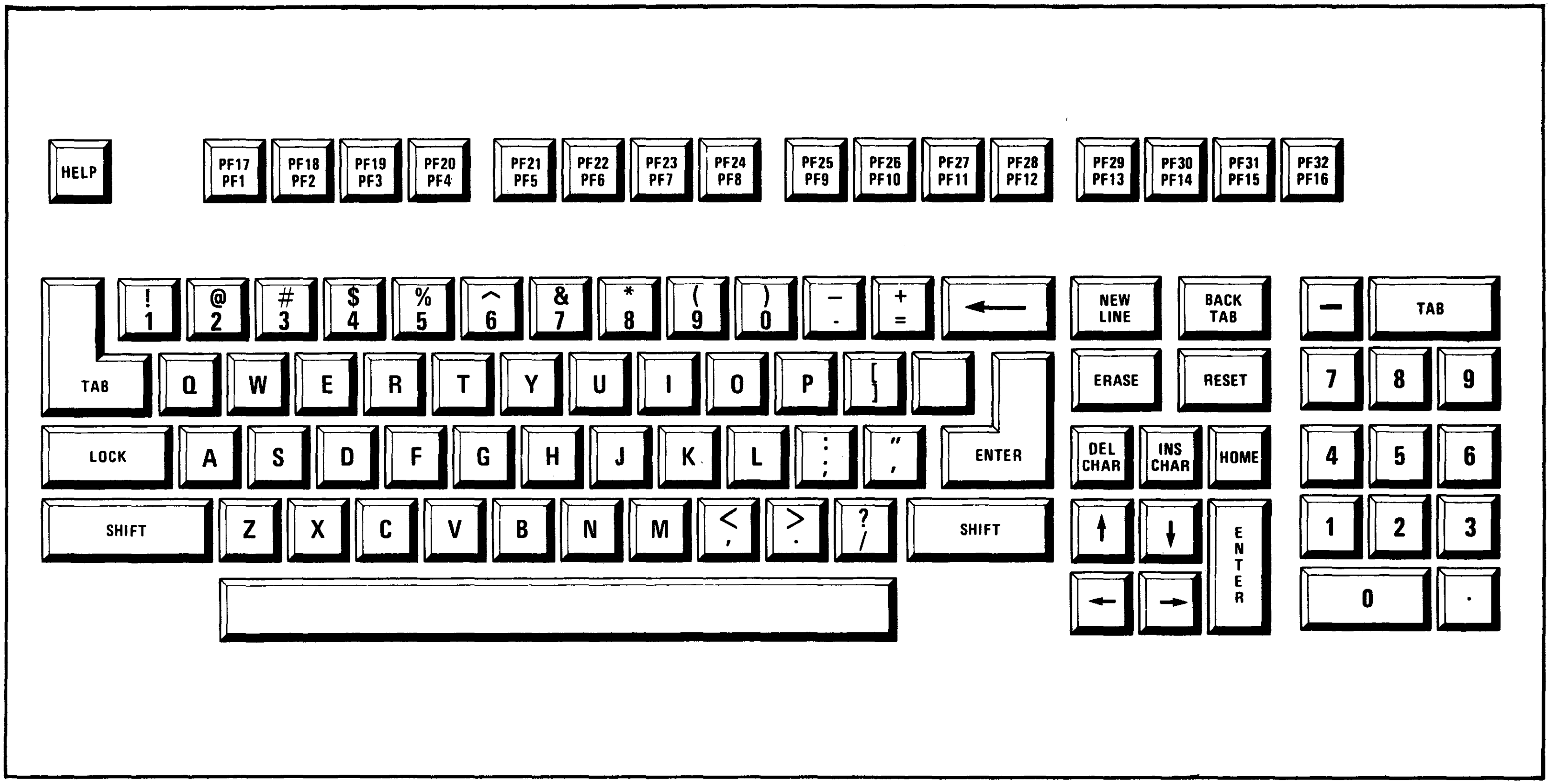


Figure 9-1. The Keyboard

Field-Sensitive Cursor Positioning Keys

The following keys normally move the cursor two or more positions after the key is pressed only once. These keys are used to move the cursor to the start of a field or a new line and can be used to simplify data entry. They position the cursor to a modifiable position.

These four keys are sensitive to modifiable positions, although none of them modify any position. The four keys of this set are as follows:

TAB - Moves the cursor to the next position within a modifiable field or to a protected numeric-only field. If there are no more modifiable positions, the alarm sounds and the cursor does not move.

BACK TAB - Positions the cursor at the first byte of the nearest modifiable 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, the alarm sounds and the cursor does not move.

NEW LINE - Advances the cursor to the first position of the next line, and then moves the cursor to the first modifiable position 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 location following the start of the next line, the alarm sounds and the cursor does not move.

HOME - Positions the cursor at the first modifiable location on the screen. If there is no modifiable location on the screen, the alarm sounds and the cursor does not move.

9.3.2 Data Entry Keys

None of the keys talked about 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, the same action as for a protected field is taken. 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 preceding the cursor are not changed. The cursor does not move.

INS (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 or a pseudoblank, is lost. If the last character in the field is not a blank or a pseudoblank, no screen location is changed, the alarm sounds, and the cursor does not move. Pseudoblanks are the characters X'0B' and X'05' in a modifiable field.

DEL (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 INS key.

9.3.3 Special Keys

SHIFT - Has the same effect as SHIFT 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.

RESET - 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.

9.3.4 Keys Communicating 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 will be set to the character for the struck key and an interruption will be presented to the computer. After these keys are pressed, all keys except the HELP key and the RESET 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 only time the key cannot be honored is when an unsolicited interruption is pending for the same device. At any other time the key is honored. This includes 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 ENTER, the only difference being in the AID byte generated.

ENTER - 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 ENTER key. The ENTER key is not honored when the keyboard is locked for data entry keys.

9.4 WORKSTATION IOCW AND I/O COMMANDS

The computer communicates with the workstation by using commands and orders. Commands are VS I/O requests specified by an SIO instruction and are part of the IOCW. Orders are requests for actions sent to the workstation as part of the data area. Workstation orders are discussed in Subsection 9.5.1; computer commands are discussed at the end of that subsection.

The following is a description of the specific interpretations of the IOCW fields for the workstation. For a general explanation, refer to the general I/O description in Chapter 8. The workstation does not have a device-dependent extension to the IOCW.

9.4.1 Command and Modifier Bits

For the valid commands and modifier bits, refer to Subsection 9.5.13.

9.4.2 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. The data to be read or written is to immediately follow the order area and is called the mapping area. The data area specified by the IOCW data address is discussed in Section 9.5 and is outlined in Table 9-3.

9.4.3 Data Count

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 will be terminated with an indication in the IOSW of incorrect length. The length of the mapping area is the data count minus the length of the order area. The length of the mapping area must not extend past the end of the workstation screen, so the maximum length of the mapping area is 1920 bytes. If the length does extend past the end of the screen, the command will be terminated with an indication of incorrect length stored in the IOSW.

9.5 DATA AREA

The data area specified in the data address for a workstation will consist of two adjacent areas: the order area and the mapping area. The order area contains the starting row number and specifications of actions to be performed (on a WRITE), or is set by the device to the cursor address (on a READ). The mapping area is the data transmitted to or from the screen and contains field attribute characters and display characters. The length of the entire data area is given by the data count field, shown in Figure 9-2.

Field	Command code	Data address	Data count field
Bit	0 7	8 31	32 47
Digit	1 2	3 4 5 6 7 8	9 10 11 12

Figure 9-2. Workstation IOCW

9.5.1 Order Area

The order area is always four bytes long. Table 9-3 shows the layout of this area.

Table 9-3. Significance of Bytes in the Workstation Order Area

Byte	On READ	On WRITE
0	Row number	Row number
1	Reserved (must be 0, except for remote workstations, where this is the AID character. Refer to Table 9-6.)	WCC (write control character; discussed below)
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.

9.5.2 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 will be 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.

9.5.3 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 will be 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 "set cursor address" 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 "set cursor address" bit is not set in the WCC, the third and fourth bytes of the order area are ignored.

If the "set cursor address" bit is set in the WCC, the cursor row address byte must be set to a value between 1 and 24 inclusive, and the cursor column address byte must be set to a value between 1 and 80 inclusive. After the WRITE completes, the cursor will be 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 will act 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 will be positioned to the first location in the specified row.

If the "set cursor address" 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 will be terminated with an indication of Order Check (OR) stored in the IOSW.

9.5.4 WCC Area

The computer controls the workstation by use of orders. An order is a request for workstation action that is coded in the order area. The order area is the first 4 bytes of the data area pointed to by the IOCW. The second byte of this area is a Write Control Character (WCC). The area also contains the starting screen location for data transfer. The positions immediately following the order area are the mapping area. Data is transferred between the mapping area and the screen as specified by the IOCW command code. The length of the mapping area is defined by the length specified in the data count in the IOCW minus 4.

WCC (write control character) - This is the second byte of every WRITE. No other byte will be interpreted as a WCC. For the interpretation of the bits of the WCC, refer to Table 9-4.

Table 9-4. 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)

9.5.5 Unlock the Keyboard

After the record is written to the screen and after sounding of the alarm, if specified, the AID character will be set to blank and the keyboard will then be unlocked.

If bit 0 is 0, the keyboard will be locked before any data is transmitted to the workstation. This can lock an unlocked keyboard. If the keyboard is locked, this bit will not change the status of the keyboard. The normal method for locking the keyboard is to wait for the operator to strike one of the computer communication keys. If the bit is 0 and the keyboard is locked, the AID character in the IOSW will not change. However, if the command locks the keyboard, the AID character will be set to " ' ".

9.5.6 Sound the Alarm

If this bit is set to 1, the alarm will sound before the data is transmitted to the screen.

9.5.7 Position the Cursor

If this bit is set to 1, after data is transferred to the screen the cursor will be positioned as described in Subsection 9.5.3.

9.5.8 Roll Down

Setting this bit 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.

9.5.9 Roll Up

If this bit is set to 1, the row specified in the order area will be lost and each line below it will be copied into the next higher line (e.g., line 1 will be replaced by the contents of line 2, etc.). This copying will proceed until the last row of the screen has been copied. The last row will then be set to blanks, and the WRITE will proceed on the last line of the screen. An attempt to write more than one line in a single command with "roll up" specified will result in Order Check (OR) being reported.

9.5.10 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.

9.5.11 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 X'8C' before the data is transferred to the screen. Therefore, there are no modifiable locations after the data that is written.

The options of the WCC on a WRITE will take place in the following order if they are chosen:

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.

9.5.12 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 will always correspond to the start of a row, the only restriction on the end of the mapping area is that it not extend past screen end. This means that the mapping area can include more than one row. No mapping area need be supplied for a 4-byte READ or WRITE (order area only).

Figure 9-3 is a summary of the order area and mapping area.

```
Offset                      Description

         _______________________ ___________
        |                       |           |
   0    | Starting row number   |           |
   1    | WCC (0 for READ)      |           |    Order area
   2    | Cursor column         |           |
   3    | Cursor row            |           |
        |_______________________|___________|
        |                                   |
   4    |         .                         |
   .    |         .                         |    Mapping area
   .    |         .                         |
```

Figure 9-3. Data Area Specified by Workstation IOCW

9.5.13 Workstation I/O Commands

Commands are I/O requests issued by the computer. The SIO instruction starts the command in the IOCW. This causes a transfer of data either from or to the workstation. The data sent either from or to the workstation will contain one or more orders and (optionally) screen characters. Data sent to the workstation with any of the WRITE commands can be used to format the screen into fields, display characters, or control workstation functions like unlocking the keyboard. Data read from the workstation includes field attribute characters that are within the range of the READ, as specified in the order area. Commands should normally be issued only when the keyboard is known to be locked, unless the screen display is to remain until ENTER is pressed. If the keyboard is not locked and a command is issued, keystrokes may be lost. The valid commands are listed in Table 9-5.

Table 9-5. Workstation Commands

Command	Command and Modifier Bits	
WRITE	10	0000N0
WRITE SELECTED	10	0100N0
WRITE TABS	10	0001N0
READ	01	0000N0
READ ALTERED	01	0100N0
READ DIAG	01	0010N0
READ TABS	01	0001N0

The bits marked "N" in Table 9-5 are set to indicate indirect data addressing. When they are set, the data address portion of the IOCW addresses an Indirect Address list as described in Chapter 8. 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.

The READ command causes the contents of the screen locations corresponding 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, however, while the keyboard is unlocked, as this may cause some operator keystrokes to be lost.

If any of the characters in the range of the READ are pseudoblanks (i.e., in a modifiable field, characters with bit patterns 00001011 (the half-solid character) or 00000101), these will be converted to blanks on the screen before the data is read. When these characters are in protected fields, they are not considered to be pseudoblanks, and they will not be 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 will be 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.

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.

The READ DIAG (diagnostic read) 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.

The READ TABS command reads into memory the column numbers of all set tabs. The command transmits up to 10 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).

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.

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 identifying the fields that are written.

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 will 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).

Table 9-6 lists the AID configurations for workstations, along with the associated hexadecimal characters and graphic characters. AID characters are discussed in Subsection 9.6.3.

Table 9-6. Attention ID (AID) Configurations

AID	Hex Character (ASCII)	Graphic Character	AID	Hex Character (ASCII)	Graphic Character
Keyboard Unlocked	20	' ' (blank)	Locked by Write	21	'
ENTER key	40	@			
PF 1 key	41	A	PF 17 key	61	a
PF 2 key	42	B	PF 18 key	62	b
PF 3 key	43	C	PF 19 key	63	c
PF 4 key	44	D	PF 20 key	64	d
PF 5 key	45	E	PF 21 key	65	e
PF 6 key	46	F	PF 22 key	66	f
PF 7 key	47	G	PF 23 key	67	g
PF 8 key	48	H	PF 24 key	68	h
PF 9 key	49	I	PF 25 key	69	i
PF 10 key	4A	J	PF 26 key	6A	j
PF 11 key	4B	K	PF 27 key	6B	k
PF 12 key	4C	L	PF 28 key	6C	l
PF 13 key	4D	M	PF 29 key	6D	m
PF 14 key	4E	N	PF 30 key	6E	n
PF 15 key	4F	O	PF 31 key	6F	o
PF 16 key	50	P	PF 32 key	70	p
HELP key	30	0	Screen damage alert	3F	?

Note: for remote workstations, X'00' = Power on (for workstation types 2246R, 2246S, and 2246C), X'01' = Disconnect (2246R only), X'02' = Connect (2246R only), X'3F' = Power on (2246P only), and X'3F' = I/O error (all others).

9.6 WORKSTATION I/O STATUS WORD

For a general discussion of the IOSW, refer to Chapter 8. The general status bits and error status bits of the IOSW are set on an interruption as described in the following subsections.

9.6.1 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 ENTER, 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.

9.6.2 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 on occurrence of main memory parity error during reading of IOCA, Indirect Address list, IOCW or data. EC always set when MPE is set.
10	MAE	Set on occurrence of main memory addressing error during reading of IOCW, Indirect Address list, or data. EC always set when MAE is set.
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 on a serial workstation during an I/O operation. Never set for parallel workstations.

IOSW Bit	Mnemonic	Meaning
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.
14	PP	Set when microprogram loading for a programmable IOP (data link processor) is required.
15	DP	Set when a serial programmable workstation (type 2246R, 2246S, or 2246C) 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.

9.6.3 Device-Dependent Bits

The workstation uses both additional status bytes of the IOSW. They are always stored on an interrupt.

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. Refer to Table 9-6. 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 1 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 if any data was transferred to memory by this command. This bit is set on normal completion of a READ ALTERED command only.

Workstation Powered-on Indication and Screen Damage Alert

When the parallel workstation (2246P) is powered on, an unsolicited interruption becomes pending with X'3F' ('?') in the first additional status byte of the IOSW (AID character position). This indicates that the screen contents, tab positions, and other previous workstation status have been lost and that the workstation screen has been filled with X'8C' bytes (default field attribute characters). In addition, the keyboard is locked and tab positions are cleared.

In the event of an I/O error resulting in completion with bits MPE or DM set (in conjunction with bit EC), X'3F' ('?') is stored in the first additional status byte (AID byte) of the IOSW.

When the serial programmable workstation (2246S) is powered on, an unsolicited interruption becomes pending with the programmable device status stored in the second extended status byte of the IOSW. The corresponding microprogram should be loaded by the CPU and restarted before normal I/O operations can be attempted.

9.7 EXAMPLE OF COMPUTER CONVERSATION WITH A WORKSTATION

When the operator powers on the workstation, an attention interrupt is generated. The system then issues a WRITE. The data transmitted to the workstation formats the screen into fields and displays the information telling 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. When the operator has finished entering data, he presses the ENTER key (or one of the other communication keys). This causes an interrupt and locks the data entry keys, program function keys, and ENTER 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 10
WANG PRINTER CHARACTERISTICS

10.1 INTRODUCTION

A number of line printers are available as optional peripherals on the Wang VS system. More than one printer may be attached to a system, the only restriction being on the total number of attached devices. Characteristics of the various printer models are shown in Table 10-1.

Table 10-1. Characteristics of Printer Models

Model	Speed	Type	Char. Set 0	Channels	Expand	Lines per Inch	Chars. per Line
2221V	200 cps	Matrix	96	1,5	Yes	6	132
2231V-2	120 cps	Matrix	96	1,5	Yes	6	132
2273V-1	250 lpm	Band	64/96	1,5	No	6/8	132/158
5521	200 cps	Matrix	96	1.5	Yes	6	132
5531-2	120 cps	Matrix	112	1,5	Yes	6/8	132
5570	600 lpm	Line	64	1-12	No	6/8	132
5571	430 lpm	Line	96	1-12	No	6/8	132
5573	250 lpm	Band	64/96	1,5	No	6/8	132/158
5574	600 lpm	Band	64/96	1-12	No	6/8	132
6581W	30 cps	Daisy	86	1,5	No	3,4,6,8	132/158
6581WC	30 cps	Daisy	86	1,5	No	3,4,6,8	180/216

10.2 PRINTER IOCW AND I/O COMMANDS

Program control of the printer is always by means of an I/O Command Word (IOCW) specifying the functions to be performed. The printer IOCW is diagrammed in Figure 10-1. For a general discussion of the IOCW, refer to Chapter 8. The first byte of the IOCW contains the I/O command and the command modifier bits. These are as follows:

```
CCMM MMMM
```

where CC is the command, and MM MMMM are the command modifier bits. Figure 10-1 summarizes the IOCW as discussed in the following subsections of section 10.2.

10.2.1 WRITE Command

The command for the printer is a WRITE command, with command code of binary 10. Bit 7 (last modifier bit) of the command byte is set to 0 to request uppercase printing, and to 1 to print lowercase letters. The other modifier bits are ignored.

10.2.2 Data Count and Data Area

The data count field in the IOCW can have values between 6 and 2048, inclusive. A data count of less than 6 or greater than 2048 results in the command's being suppressed with error and incorrect length indications. The data area as addressed by the IOCW is as follows:

```
 ___________________________    ___________________
|    |              | / /    |                  |
| BL | Data record  | var    | Data record      |
|____|______________|/_/_____|__________________|

 0    2                                      ...
```

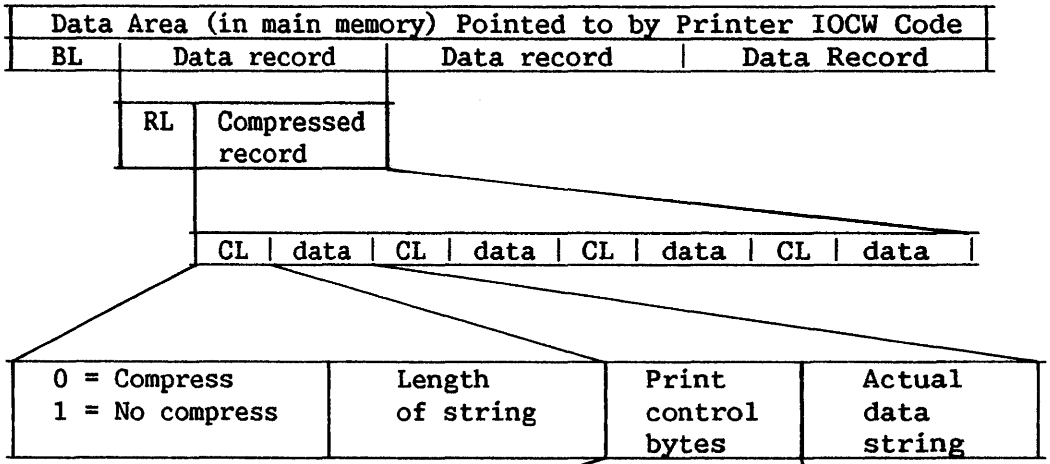
where BL is the total area (block) length in binary, including the BL bytes. The 2-byte field BL must contain the same value as the IOCW data count field, or the command is suppressed with error and incorrect length indications. Each data record is as follows:

```
 ______________________________
|    |                        |
| RL | Compressed record      |
|____|________________________|
 0    2                    var
```

where RL is the compressed record length in binary, including the RL bytes.

Field	Command code	Data address	Data count field
Bit	0 7	8 31	32 47
Digit	1 2	3 4 5 6 7 8	9 10 11 12

Command code	Data address	Data count field
80 = WRITE	Physical address in main memory	Number of bytes to be transferred



Print Control Bytes
(First two data bytes)

Byte	Bit	Meaning
0	0	Line or channel spacing select 0 = Space lines as specified in the second byte 1 = Skip to the channel specified in the second byte
	1	0 = Space before printing 1 = Space after printing
	2	0 = Normal width characters 1 Double width characters
	3	1 = Activate hardware alarm
	4-7	Reserved
1	0	Reserved
	11-17	Binary number of lines to space (0-127), or Channel for skip (1-12)

Figure 10-1. Printer IOCW Format

The compressed record, which includes print control bytes, is in the format produced by the COMP instruction. When expanded, it must be not less than two bytes in length or the command is terminated with error and incorrect length indications. (Lengths greater than the form width result in right-truncation on some printer models.) A compressed record is as follows:

```
 ___________________/ /______________________
|    |        | / /     |    |        |
| CL | data   | var     | CL | data   |
|____|________|/ /______|____|________|
 0    1                       ...
```

where CL indicates a compression length byte as for the COMPRESS STRING instruction, and is followed by one or more data bytes. If a data area extends beyond the end of the record as determined by length RL, the command is terminated with error and incorrect length indications.

10.2.3 Print Control Bytes

The bits of the control bytes are set to specify all control operations for each printer WRITE. The first two (expanded) data bytes of a record are print control bytes, defined as indicated in Table 10-2.

Table 10-2. Printer Control Codes

Control		
Byte	Bit	Function
0	0	Line or channel spacing select: 0 = Space number of lines specified in second control byte 1 = Skip to the channel specified in the second byte
	1	0 = Space before printing 1 = Space after printing
	2	0 = Normal width characters 1 = Double width characters
	3	1 = Activate hardware alarm
	4-7	Reserved
1	0	Reserved
	1-7	Binary number of lines to space (0-127), or channel for skip (1-12)

The control bytes are designed for all printers. The following restrictions apply to the currently available printers:

- Suppress spacing (space 0) is valid.
- Only skips to channel 1 (top of form) and channel 5 (vertical tab) are valid for the current matrix and daisy printers.
- Intermixed pre-spacing and post-spacing produce correct results.
- Invalid control bytes cause an automatic single space after print.

A valid WRITE command causes preprint controls, as well as character width setting and hardware alarm status, to be interpreted and executed by the printer.

Nonprintable characters print as backslashes (\) or blanks, depending upon the printers.

Successive bytes from the character area are sent to the printer buffer and the IOCW count is decremented until it reaches 0, at which time the operation normally is completed.

10.3 PRINTER I/O STATUS WORD

When applicable, an "out-of-paper," "ribbon-out," "cover-open," or "deselect" condition is reported as an interruption if an operation is attempted on a printer that has only the intervention required (IRQ) bit set. The operation is halted at the beginning of the next line of printing or spacing. When the printer is again successfully selected, the operation continues normally. Parallel printers also report a "power-off" condition as an Intervention Required (IRQ) interrupt. But for serial printers (denoted by a final letter S in their model numbers), a "power-off" condition is reported as an error completion, with power-off status stored in the extended status byte 2. The microprogram must then be reloaded before normal I/O operation can be resumed.

For both parallel and serial printers, part or all of a line or block of lines may be lost if a power-off condition occurs during an I/O operation.

The general status bits and error status bits of the IOSW are set on an interrupt as described in the following subsections.

10.3.1 General Status Byte

IOSW Bit	Mnemonic	Meaning
0	IRQ	Set on device-deselect, ribbon-out, cover-open, or out-of-paper condition only; thus, only while a WRITE is outstanding. NC or EC never set with IRQ (see preceding discussion). PC may be set with IRQ.
1	NC	Set on normal completion (see next section).
2	EC	Set on completion with error. NC and EC never set together.
3	U	Set when a serial printer is powered on.
4	PC	Set only in conjunction with IRQ, NC, or EC.
5	DAR	Set for some printers on programmable I/O processors in advance of the completion interruption for a WRITE operation. Indicates that the main memory data area containing the block of records to be printed will not be re-accessed and may be reused or erased by the central processor program. No other status bits are set with DAR.
6-7		Reserved (always 0)

10.3.2 Error Status Byte

IOSW Bit	Mnemonic	Meaning
8	IC	Set to indicate invalid command byte in IOCW, IOCW not fullword aligned, or data area not fullword aligned. EC always set when IC is set.
9	MPE	Set on occurrence of main memory parity error during reading of IOCA, IOCW, or data. EC always set when MPE is set.
10	MAE	Set on occurrence of main memory addressing error during reading of IOCW or data. EC always set when MAE is set.
11-12	DM,DAM	Never set.
13	IL	Set if IOCW specifies an invalid data count, if the IOCW data count is exhausted before end of block is reached, or if a record length byte is invalid.

IOSW Bit	Mnemonic	Meaning
14	PP	Set when microprogram loading for a programmable IOP (data link processor) is required.
15	DP	Set when the serial programmable workstation is powered off or 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.

10.3.3 Device-Dependent and Residual Count Bytes

The IOSW device-dependent status field (third and fourth bytes of the IOSW) contains a count of the number of lines printed on any completion (including completion after HIO). The fifth and sixth bytes contain the residual count for the operation, in bytes.

10.3.4 HALT I/O to Printer

The printers accept HALT I/O (HIO) instructions as defined in Chapter 7. If condition code 1 results from an HIO instruction, a completion interruption (bit NC of the IOSW set) will become pending when the printer has been cleared of any outstanding operation and of any pending interruptions.

CHAPTER 11
WANG DISK FACILITY CHARACTERISTICS

11.1 INTRODUCTION

The VS provides several different disk IOPs to support the various disk drives available. The 22V02 IOP supports a 2270V diskette drive and up to three 2260V 10-megabyte disk drives, or three 2260V 10-megabyte disk drives. The 22V08 IOP supports up to four 2265V-1, 2265V-2, 2280V-1, 2280V-2, or 2280V-3 disk drives.

The 2260V has a fixed and a removable platter, while the 2265V-1 and 2265V-2 both have a removable-only multiplatter disk pack. The 2270V has only a removable platter. The 2280V-1, 2280V-2, and 2280V-3 each have a 15M-byte removable platter, with the remaining storage area fixed. Disk drive specifications are summarized in Table 11-1.

Table 11-1. Characteristics of Disk Drive Models

Disk	2260V	2265V-1	2265V-2	2270V	2280V-1	2280V-2	2280V-3
Removable platter/pack	Platter	Pack	Pack	Platter	Platter	Platter	Platter
Tracks per cylinder	4	5	19	1	1+1	1+3	1+5
Cylinders	408	823	823	77	823	823	823
Sector size (in bytes)	256	2048	2048	256	2048	2048	2048
Sectors per track	24	9	9	16	9	9	9
Total storage (in millions of bytes)	10.03	75.85	288.22	.3154	30.34	60.68	91.02
Seek average (in ms)	38	30	30	424	30	30	30
Seek max (in ms)	130	55	55	847	55	55	55
Seek min (in ms)	9	6	6	11	6	6	6
Full rotation time (in ms)	25	16.66	16.66	167	16.66	16.66	16.66
Data transfer rate (in bytes/sec)	312K	1.2M	1.2M	31K	1.2M	1.2M	1.2M

The 2260V disk drive consists of two disk platters, one permanently fixed and the other removable. Each platter has two disk surfaces. The capacity of each platter is approximately 5.01 million bytes of data, formatted into two surfaces of 408 tracks each. A cylinder consists of all tracks that have a common access-arm position and that can be addressed without moving the access arm.

The 2265V-1 and V-2 facilities consist of one removable disk pack containing 5 usable recording surfaces (for the V-1) or 19 usable recording surfaces (for the V-2). One access arm controls the positioning on all surfaces. Each recording surface is formatted into 823 tracks, with each track divided into 9 sectors containing 2048 bytes of data apiece.

The 2270V diskette drive consists of one diskette and one access arm. Only one surface of the diskette contains data. In order to minimize wear on the diskette, the IOP unloads the access arm after a period of inactivity lasting longer than 0.6 second. There is a 30- to 50-ms delay associated with the loading of the head. The total capacity of the diskette is 315,392 bytes, formatted into 77 tracks of 16 sectors each. Each sector contains 256 bytes of data.

The 2280V-1, V-2, and V-3 disk units consist of one removable platter and one (V-1), two (V-2), or three (V-3) fixed platters. The removable platter contains one usable recording surface. The fixed platters have one (V-1), three (V-2), or five (V-3) usable recording surfaces. One access arm controls the positioning on all surfaces. Each surface is formatted into 823 tracks of 9 sectors apiece, each sector containing 2048 bytes of data.

The addressing scheme for the 2260V provides relative sector addressing of 256-byte sectors for these units. The last three bytes of the IOCW contain the sector address. The first of these three bytes must be 0 or the command will be rejected with an indication of invalid command and invalid disk address. Both surfaces of a platter are used for recording. On track 0, the lower surface contains sectors 0 through 23; the upper surface contains sectors 24 through 47. The lower surface of Track 1 contains sectors 48 through 71; the upper surface contains sectors 72 through 95, etc.

The 2270V provides the same addressing scheme except that only one surface is used for recording and each track contains 16 sectors. Sectors 0 through 15 are in track 0; sectors 16 through 31 are in track 1, etc.

The addressing scheme for the 2265V-1, 2265V-2, 2280V-1, 2280V-2, and 2280V-3 provides relative sector addressing of 2048-byte sectors for these units. The last three bytes of the IOCW contain the sector address times 8. (Thus the lowest three bits must be 0s.) Each surface contains nine 2048-byte sectors. On the 2265V-1, 45 consecutive sectors can be addressed without access-arm movement. On the 2265V-2, 171 sectors can be addressed without access-arm movement.

The platter to be used is specified in the modifier bits of the command code of the IOCW. These bits are ignored for the 2265V-1, 2265V-2, and 2270V facilities.

The number of sectors to be transferred should not be greater than the number of sectors in a cylinder. Head select change is accomplished during the intersector time. This permits a full track of data to be transferred in one rotation, and a full cylinder of data in a minimum number of rotations. A multisector transfer need not start or end on a track boundary and may span more than one track of a cylinder. 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.

11.2 DISK IOCW AND I/O COMMANDS

I/O commands to the disk consist of a command, command modifier bits, a memory address, a data count, and a disk address (device-dependent section).

The I/O Command Word (IOCW) is discussed in Chapter 8. The first byte of the IOCW contains the I/O command and the command modifier bits. These are as follows:

CCCMMMMM

where CCC is the command and MMMMM are the command modifier bits. Valid commands are listed in Table 11-2.

Table 11-2. Valid I/O Commands

Command Code	Command Function
010X XXXX	READ disk sector(s)
100X XXXX	WRITE disk sector(s)
101X XXXX	WRITE VERIFY
110X XXXX	SEEK
111X XXXX	FORMAT

If the command byte is not as defined, the command will be rejected with an indication of invalid command.

The command modifier bits are presented in the following listing.

Command Modifier Bit	Modifier Function
0X000	Read/Write diagnostics for 2265V-1, 2265V-2, 2280V-1, 2280V-2, and 2280V-3. When this bit is set, the data transfer is limited to one sector only. Indirect data addressing is not allowed. When this bit is set on a READ command, the eight bytes of ECC code associated with the specified sector are transferred into memory immediately following the data. The total data transferred is (2048 + 8) bytes. ECC error recovery is suppressed.

Command Modifier Bit   Modifier Function

0X000 (continued)   When this bit is set with a WRITE command, 2048 bytes of memory, starting from the address specified in the IOCW, are transferred to the disk sector as data, as in a normal READ command. However, instead of generating an ECC code for the sector, the disk hardware uses the next 8 bytes of memory as its ECC code and writes these into the sector. ECC error recovery is suppressed.

00X00   Suppress retry. When this bit is set, 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).

000X0   Indirect data addressing. When this bit is set, the data address portion of the IOCW addresses an Indirect Address list as described in Chapter 8. For disk devices, the address contained in the first entry of the Indirect Address list is required to have 11 low-order 0s (2048-byte alignment).

0000X   Removable platter operation. When this bit is on, the I/O command disk address refers to the removable platter. Otherwise the command is for the fixed platter. (Ignored for the 2265V-1, 2265V-2, and 2270V facilities.)

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 sector size (256 or 2048), or the command will be rejected with an indication of "invalid data count." The sector address field of the IOCW specifies the starting sector on the disk to which the command pertains. As for all I/O, the memory address must be fullword aligned. If indirect data addressing is specified, the memory data area (as addressed by the first word of the Indirect Address list) must be 256-byte aligned for the 2260V and 2270V, and 2048-byte aligned for the 2265V-1, 2265V-2, 2280V-1, 2280V-2, and 2280V-3. If these conditions are not met, the command will be rejected with an indication of "invalid command."

11.2.1 READ Command

Execution of a READ command results in the positioning of the access mechanism and the transfer of information from the disk into memory. On all READ commands, the access mechanism is positioned at the specified sector within the specified cylinder. On all READ commands, the disk verifies (using the cyclic redundancy checks 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, the I/O device will initiate 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 (status word) with both NC and EC bits set. The appropriate status bits are set in the IOSW to describe the error.

If an irrecoverable error occurs (i.e., if all retry attempts have failed), then as much data as possible is transferred to memory and the error is reported as irrecoverable by means of the IOSW.

The READ command allows any number of sectors on a cylinder to be read by one command.

11.2.2 WRITE and WRITE (VERIFY) Commands

Execution of a WRITE command results in transfer of information from memory to the disk. On all WRITE commands, the access mechanism is positioned to the specified cylinder and sector within the cylinder. As the WRITE operation is performed, a Cyclic Redundancy Check (CRC) for parity is computed by the disk controller and appended to the sector record. For the WRITE (VERIFY) command on the 2260V or 2270V, 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. The sector addresses and the CRC are also verified. On the 2265V-1, 2265V-2, 2280V-1, 2280V-2, and 2280V-3, the ECC is recalculated and checked; comparison of data with memory does not occur. This form of the WRITE command requires a second revolution of the disk mechanism. If an error is found during data verification and retry attempts fail, the residual data count in the stored IOSW indicates the sector in error. This residual count is decremented each time a sector is successfully verified. Thus an 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 256 or 2048, and so forth.

If any error occurs during a WRITE command, retry procedures are automatically instituted as for READ. If retry is successful, the operation is reported back through the IOSW with both Normal Completion (NC) and Error Completion (EC) bits set. As with READ, in cases of hard (irrecoverable) errors the disk operation is terminated and must be restarted by another SIO operation, if desired.

The WRITE command allows any number of sectors on a cylinder to be written.

11.3 DISK CONTROL COMMANDS

11.3.1 SEEK

Execution of a SEEK disk address command results in a positioning of the disk access mechanism. The block 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.

11.3.2 FORMAT

The FORMAT command will cause the addressed sectors to be formatted with sector preambles. The data in the sectors after successful execution of the command is unpredictable.

The FORMAT command allows any number of sectors on a cylinder to be formatted.

11.4 DISK I/O STATUS WORD

The IOSW is presented in Chapter 8. The general status bits and error status bits of the IOSW are set on an interruption as described in the following subsections.

11.4.1 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 corrected by retry. NC and EC both set if a retry is successful.
3	U	Set whenever device becomes ready after the RUN button at the device is pressed. NC, EC, or PC may also be set.
4	PC	Set only with NC, EC, or U.
5-6		Reserved (always 0).
7		Reserved for software use.

11.4.2 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 not properly aligned (fullword for non-indirect-addressing operation, and 256-byte for indirect-addressing operation). EC always set if IC is set.

IOSW		
Bit	Mnemonic	Meaning
9	MPE	Set on occurrence of main memory parity error during reading of IOCA, IOCW, Indirect Address list, or data. EC always set if MPE is set.
10	MAE	Set on occurrence of main memory addressing error during reading of IOCW, Indirect Address list, or data, or writing data. EC always set if MAE is set.
11	DM	Set if and 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 . Short sector.
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).

11.4.3 Device-Dependent Bits

In addition to the general device status bits, the following extended status bits or device-dependent section in the IOSW refers to specific disk and/or controller states. These bits are discussed in detail below.

IOSW		
Bit	Mnemonic	Meaning
16		Sector reformatted on WRITE retry
17		Sector leader skipped on READ retry (except 2260V and 2270V)
18		ECC transferred (except 2260V and 2270V)
19		Reserved
20	IDA	1 = Invalid disk address
21	IDC	1 = Invalid data count

IOSW Bit	Mnemonic	Meaning
22	SO	1 = Sector overrun
23	SI	1 = SEEK incomplete
24	WP	1 = Write protect
25	NRO	1 = Not ready during operation
26	ST	1 = Timeout on sector
27	DC	1 = Data compare error
28	IID	1 = Invalid sector ID
29	CRC	1 = Invalid CRC or ECC
30	O	1 = Overrun
31	ISP	1 = Short sector
32-47		Residual data count

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 sector size.
- Data count is greater than the number of data bytes in a cylinder.
- Data count implies a cylinder change.
- Data count is not equal to the number of data bytes in a track surface for the FORMAT command.
- 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 if the disk was unable to complete a SEEK due to a malfunction. Whenever this situation arises, the access arms are placed at cylinder 0 and the SEEK is retried. This bit is not set by the 2270V.

Write Protect

This bit is set to 1 only for the 2270V on all commands when the diskette is write protected. The setting of this bit on a WRITE command causes the error completion bit and the invalid command bit to be set to 1.

Not Ready During Operation

This bit is set to 1 whenever a disk becomes not ready during an operation. The setting of this bit causes DEVICE MALFUNCTION to be set and a later DEVICE NOW READY to be reported.

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. When this situation arises, the access arms are placed at cylinder 0 and the command is retried.

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.

Short Sector

This bit is set to 1 when the controller detects the end of a sector before the operation for that sector has come to an end. The most likely cause of this error is lack of formatting.

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

For all disks except 2260V and 2270V, whenever an error is detected that is reflected in the device-dependent status and that is not a specification error (invalid disk address or invalid data count), the retry count (bits 52-55) is updated. After four retries of a particular type, the retry indicators (bits 48-51) are updated, the retry count is zeroed, and the next type of retry is attempted.

For the 2260V and 2270V, the retry indicators (bits 48-55) will be updated after up to 16 retries, with no special adjustment for both READ and WRITE operations.

Diskette Drive Indicator Bits

IOSW Bit	Value	Meaning
56	= 1	Diskette write protected
60	= 1	Soft-sectored medium
61	= 1	Double-sided medium (valid only for soft-sectored diskettes)

11.4.4 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

The disk IOPs do not support Halt I/O. 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 12
WANG MAGNETIC TAPE CHARACTERISTICS

12.1 INTRODUCTION

The Wang 2209V-1, 2209V-2, and 2209V-3 magnetic tape drives are optional features of the Wang VS. More than one tape drive may be attached to a system, restricted only by the total number of attached devices. The 2209V-1 is a 9-track 1/2-inch (13-mm) tape transport and controller that records at 1600 bits per inch (bpi) (phase encoded). The 2209V-2 is a 9-track 1/2-inch tape transport and controller that records at 1600 bpi (phase encoded) and 800 bpi (NRZI). The 2209V-3 is a 7-track 1/2-inch tape transport which records at 800 bpi (NRZI). The tape drives and controllers are compatible with industry standards.

The magnetic tape is designed to facilitate information interchange between the Wang VS and other computer systems.

12.2 MAGNETIC TAPE GENERAL DESCRIPTION

12.2.1 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 representing 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 12-1, numbering the tracks 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 12-1. Tape Bit Positions

The 7-track tape bit positions 0-5 correspond to main memory locations 2-7.

12.2.2 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.

12.2.3 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 1/32 inch (0.75 mm) from the edge of the tape nearest the tape unit.

12.2.4 End-of-Tape Marker

About 14 feet (4 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 1/32 inch (0.75 mm) 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. A WRITE or WRITE TAPE MARK operation into this area sets the EOT indicator (bit 29 of the IOSW).

12.2.5 File Protection

Because the writing operation automatically erases any previous information on the tape, 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 to enable the machine to write on the tape in the reel. When the ring is removed, only reading can take place; the file is thus protected from accidental writing, which could erase valuable information.

12.2.6 Tape Blocks

Information on tape is arranged in blocks, as shown in Figure 12-2. A tape block consists of one or more records, which are logical units of data. Blocks are separated on tape by an interblock gap--a length of blank tape of approximately 0.6 inch (15 mm). 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.

```
                    Forward Tape Motion ------>

   _____________________________________________________________________
   |//////|   |  |  |         |         |         |         |    |//////|
   |//////|IB |L | C|Logical  |Logical  |Logical  |Logical  |IB  |//////|
   |//////|gap|R | R|data     |data     |data     |data     |gap |//////|
   |//////|   |C | C|record   |record   |record   |record   |    |//////|

         |-------------------one block--------------------|

               IB Gap - Interblock gap
                  LRC - Longitudinal redundancy check
                  CRC - Cyclic redundancy check
```

Figure 12-2. Tape Blocks

12.2.7 Tape Mark

The end of a file of information is indicated by a tape mark--a special block written only by a WRITE TAPE MARK command. The tape mark is a special single-byte block with X'13' (X'OF' for 7-track tape). One or more files may be written on a reel of tape.

12.2.8 Checking Tape Validity

The validity of information written on or read from tape is insured 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.

12.3 TAPE IOCW AND I/O COMMANDS

The IOCW is discussed in Chapter 8. I/O commands to the magnetic tape consist of a command, a memory address, and a data count, as shown in Figure 12-3. There is no device-dependent section.

Command (1 byte)	Memory address (3 bytes)	Data count (2 bytes)

Figure 12-3. Tape IOCW

The first byte of the IOCW contains the I/O command, the command modifier bits, the indirect data addressing flag, and the suppress length indication flags. 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

If the reduced retry flag is set for a READ command, the number of attempts to read a tape block before reporting a hard error will be reduced from 100 to 5.

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. The data count portion of the IOCW contains the number of bytes to be transferred.

12.3.1 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.

Reading a tape mark sets the TM indicator (bit 28 of the IOSW). The tape mark is not sent to storage. Sensing the EOT reflective marker during the read operation does not cause any status to be indicated.

12.3.2 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 appropriate. Sensing the EOT reflective marker during a WRITE operation sets the EOT indicator in the IOSW.

12.4 TAPE CONTROL COMMANDS

The control operations, specified by the modifier bits of the command code, involve no data transfer. With the exceptions of SENSE, SET DENSITY, and SET PARITY, they are tape motion commands. A control command ignores the memory address and the data count fields of the IOCW. The command modifier bits correspond to the functions listed in Table 12-1.

Table 12-1. Tape Control Modifier Bits

Command Modifier Bits	Command Control Function
0000	SENSE
0100	ERASE TAPE
0101	WRITE TAPE MARK
0110	FORWARD SPACE BLOCK
0111	FORWARD SPACE FILE
1000	REWIND
1001	REWIND/UNLOAD
1010	BACKSPACE BLOCK
1011	BACKSPACE FILE
1100	SET DENSITY to 1600 bpi (phase encoded)
1101	SET DENSITY to 800 bpi (NRZI)
1110	SET PARITY to odd (7-track only)
1111	SET PARITY to even (7-track only)

A valid control command causes the specified control function to be executed. At the end of the control function, a normal completion interruption is issued.

12.4.1 ERASE TAPE

Execution of an ERASE TAPE command causes the tape drive to erase approximately 3.75 inches (9.5 cm) of tape. Performing this operation in the EOT area sets the EOT indicator in the IOSW.

12.4.2 REWIND

Execution of a REWIND control command causes the tape drive to enter rewind mode and reposition the tape at the LP reflective marker. This causes the LP indicator to be set in the IOSW.

12.4.3 REWIND AND UNLOAD

Execution of a REWIND/UNLOAD control command causes the tape drive to enter rewind mode, reposition the tape at the LP reflective marker, and reset the tape status to off line.

12.4.4 WRITE TAPE MARK

Execution of the WRITE TAPE MARK command causes the tape drive to write a tape mark (a special block) on tape. Performing this operation in the EOT area sets the EOT indicator in the IOSW.

12.4.5 FORWARD SPACE BLOCK

Execution of the FORWARD SPACE BLOCK command causes the tape drive to move the tape forward to the next interblock gap. Sensing a tape mark sets the TM indicator in the IOSW.

12.4.6 FORWARD SPACE FILE

Execution of the FORWARD SPACE FILE command causes the tape drive to move 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.

12.4.7 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).

12.4.8 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. Sensing the tape mark does not cause the TM indicator in the IOSW to be set.

12.4.9 SENSE

Execution of the SENSE command causes the current status of the tape drive to be stored in the IOSW.

12.4.10 SET DENSITY

Execution of the SET DENSITY command causes the tape drive to be logically switched 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 for 9-track drives only, and effective only for dual-density drives.

12.4.11 SET PARITY

Execution of the SET PARITY command causes the tape drive to be logically switched to the selected parity. The IOSW indicates the change in the parity status bit. This command is valid for 7-track drives only.

12.4.12 Effect of Tape Markers on IOSW Bits

The following tape markers, when encountered by the specified operation, set the LP, TM, EC, and EOT bits if those bits are listed in the corresponding row and column.

Tape Operation	Load Point Mark (LP)	Tape Mark (TM)	End-of-Tape Mark (EOT)
WRITE	-	-	EOT
READ	-	TM	-
REWIND	LP	-	-
REWIND/UNLOAD	-	-	-
WRITE TAPE MARK	-	-	EOT
FORWARD SPACE BLOCK	-	TM	-
FORWARD SPACE FILE	-	-	-
BACKSPACE BLOCK	LP (EC)	TM	-
BACKSPACE FILE	LP	-	-

Programming Note: Tape drives have no interlocking to prevent the execution of improper sequences of WRITE and READ operations that may result in writing partial blocks on tape. Avoiding these improper sequences is a program responsibility.

12.5 TAPE I/O STATUS WORD

The I/O Status Word (IOSW) is discussed in Chapter 8. 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 is manually loaded and reaches load point, an unsolicited interruption occurs with general status bit U (unsolicited interruption) set. IOSW bits LP (load point) and/or FP (file protected) may be set.

12.5.1 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. NC, EC, or PC may also be set.
4	PC	IOP now ready; may be set alone or with NC, EC, or U.
5-7		Reserved (always 0).

12.5.2 Error Status Byte

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, or data not properly aligned (i.e., not fullword aligned for non-indirect-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 on a 9-track drive; attempt made to issue diagnostic commands except on a 9-track NRZI drive, or to use data length over 32,768; attempt made to backspace when tape is at load point, or to write, write tape mark, or erase tape when tape is write protected; attempt made to specify a maximum size READ operation of less than 8 bytes, or a maximum size WRITE operation of less than 12 bytes.
9	MPE	Set in conjunction with EC on occurrence of main memory parity error reading IOCA, IOCW, Indirect Address List, or data.
10	MAE	Set in conjunction with EC on occurrence of main memory addressing error reading IOCW, Indirect Address List or data, or writing data.
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 disk 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).

12.5.3 Device-Dependent Bits

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	VRC	1 = Invalid vertical redundancy check
17	LRC	1 = Invalid longitudinal redundancy check
18	CRC	1 = Invalid cyclic redundancy check
19	SC	1 = Skew check on WRITE
20	WD	1 = Wrong density (density not available)
21	PE	1 = Phase-encoded ID burst detected
22	TR7	1 = 7-track tape drive
23	PAR	0 = Odd parity; 1 = Even parity
24	FP	1 = File protected
25-26	Density	00 = 1600 bpi (phase encoded) 10 = 800 bpi (NRZI)
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
32-47		Residual data count bytes
48-55		Error count bits

These bits are discussed in detail in the following subsections.

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

This bit is set during a WRITE operation if excessive skew is detected.

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.

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

This bit is set whenever the EOT reflective strip has been detected during a WRITE or WRITE TAPE MARK 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.

Error Count

When an error is detected (as indicated in the following extended status bits), 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 will be attempted upon the detection of an error: the tape will be backspaced over the block just written, then an ERASE operation will be initiated to erase over 3.75 inches (9.5 cm) of tape, and the WRITE operation retried. For a READ operation, a maximum of 100 retries (5 retries if bit R is set in IOCW) of the following sequence is attempted: the tape is backspaced over the block just read, and the READ operation is retried.

APPENDIX A
OPERATION CODE AND ASCII CHARACTER LIST

Op Code	Mnemonic	Format	Character
00	---	RR	
01	BCS	RR	
02	SIO	RR	
03	HIO	RR	
04	RTC	RR	
05	BALR	RR	
06	BCTR	RR	
07	BCR	RR	
08	POP	RR	
09	POPH	RR	
0A	SVC	RR	
0B	PUSH	RR	
0C	CIO	RR	
0D	SPM	RR	
0E	MVCL	RR	
0F	CLCL	RR	
10	LPR	RR	
11	LNR	RR	
12	LTR	RR	
13	LCR	RR	
14	NR	RR	
15	CLR	RR	
16	OR	RR	
17	XR	RR	
18	LR	RR	
19	CR	RR	
1A	AR	RR	
1B	SR	RR	
1C	MR	RR	
1D	DR	RR	
1E	ALR	RR	
1F	SLR	RR	
20	LPDR, LPER	RR	(Space)
21	LNDR, LNER	RR	'
22	LTDR, LTER	RR	"
23	LCDR, LCER	RR	#
24	HDR, HER	RR	$
25	LDER, LRER	RR	%
26	RPC	RR	&
27	SVCX	RR	'
28	LDR, LER	RR	(
29	CDR, CER	RR	)
2A	ADR, AER	RR	*
2B	SDR, SER	RR	+
2C	MDR, MER	RR	,
2D	DDR, DER	RR	-
2E	CID	RR	.
2F	CDI	RR	/
30			0
31			1
32			2
33			3
34			4
35			5
36			6
37			7
38			8
39			9
3A	AQR	RR	:
3B	SQR	RR	;
3C	MQR	RR	<
3D	DQR	RR	=
3E			>
3F			?
40	STH	RX	@
41	LA	RX	A
42	STC	RX	B
43	IC	RX	C
44	EX	RX	D
45	BAL	RX	E
46	BCT	RX	F
47	BC	RX	G
48	LH	RX	H
49	CH	RX	I
4A	AH	RX	J
4B	SH	RX	K
4C	MH	RX	L
4D	LT	RX	M
4E	CVD	RX	N
4F	CVB	RX	O
50	ST	RX	P
51	DSEM	RX	Q
52	ENQ	RX	R
53	ENSK	RX	S
54	N	RX	T
55	CL	RX	U

Op Code	Mnemonic	Format	Character
56	O	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	
86	BHX	RS	
87	BXLE	RS	
88	SRL	RS	
89	SRL	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	LPTO, LSCTL	RS	
A4	LPT1, STSCTL	RS	
A5	LPT2	RS	
A6	POPM	RS	
A7	LPPT	RS	
A8	LOT	RX	
A9	PUSHM	RS	
AA			
AB	Reserved for diagnostic use		
AC	STNSM	RS	
AD	STOSM	RS	
AE	SPFT	RS	
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	
CO			
C1			
C2			
C3			
C4	PAL	SS	
C5			
C6			
C7	MTQ	SS	
C8			
C9			
CA			
CB			
CC			
CD			
CE			
CF			
DO			
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	
E6			
E7			
E8			
E9			
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	LSREG	S
9B02	STSREG	S
9B03-9B7F	Unused	
9B80	STCPID	
9B81-9BFF	Unused	

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 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 7 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

Eight 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 Address (IOCA)
The address of the I/O Command Word (IOCW) to be executed for a device. The I/O Command Area reserves two bytes for the IOCA for each device.

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 (IOP)
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 Frame Table
In local memory, a table with two bits for each page frame of main memory.

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 of the three segments 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.

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.

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, LPTO, LPT1, LPT2, LPCW, LSREG, STNSM, STOSM, RRCB, SIO, STDD, STSREG, and SVCX.

Reference Bit
The reference bit is turned on by hardware whenever the associated page in real storage is referred to.

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.

Sector
That part of a track on a disk that can fit into a page (2K bytes).

Segment

An area of contiguous virtual addresses beginning on a 1,048,576-byte boundary, whose address translation is effected through a single local page table. A segment may consist of more than one page, with only portions of a segment being within main memory at any time.

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.

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 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 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.

This manual is updated by: Addendum 800-1100PO-04.01

DOCUMENT HISTORY

4TH EDITION OF VS PRINCIPLES OF OPERATION

Type	Description	Pages
New Features	Index	INDEX-1
	Glossary	B-1
	New tables and figures:	
	. Data representation and boundary alignment	3-5
	. Sample PCW	4-2
	. Sample IOCW	8-4.5
	. Sample IOSW	8-10
	. Workstation IOCW	9-6
	. Printer IOCW	10-3
	SSI instructions format	3-4
	More on the CIO instruction	8-13
	Tape general status byte and error status byte	12-7
Documentation Changes	Chapters 6 and 8 reversed	
	Chapters 9-12 reorganized to follow format of the new Chapter 8	
	The section on debugging aids moved from Chapter 3 to Chapter 5. The debugging aids have been completely redesigned.	5-8
	Section 3.1 on instruction formats reorganized	3-1
	The pages on the EDIT instruction reorganized	7-53
	"Stack pointer" substituted for "stack top word"	3-16
	Changes in instruction formats for:	
	. CLPC	7-38
	. POPC	7-119
	. PUSHC	7-126

This manual is updated by: Addendum 800-1100PO-04.01

Type	Description	Pages
Documentation Changes (continued)	The Physical Destination Trap has been removed.	
	A special 32-bit internal register, accessible only by the (privileged) instructions LSREG, STSREG, and STDD, has been created.	7-98
	Instructions deleted:	
	. BFBV	
	. FIX	
	. STS	
	. UNFIX	
	Instructions modified:	
	. CID	7-44
	. STDD	7-155
	New instructions:	
	. CLCL COMPARE LOGICAL LONG	7-36
	. LSREG LOAD SPECIAL REGISTER	7-98
	. LOT LOAD OR TRAP	7-89
	. LPPT LOAD PARTIAL PAGE TABLE	7-91
	. MTQ MODIFY TIMER QUEUE	7-178
	. MVCL MOVE CHARACTERS LONG	7-100
	. RPC RETURN AND POP ON CONDITION	7-130
	. SCAN SCAN FOR BYTE	7-134
	. STSREG STORE SPECIAL REGISTER	7-160
	. STCPID STORE CP AND MICROCODE ID	7-154
	Short floating-point instructions (new):	
	. AE, AER ADD NORMALIZED	7-7
	. AU ADD UNNORMALIZED	7-9
	. CE, CER COMPARE	7-29
	. DE, DER DIVIDE	7-49
	. HER HALVE	7-69
	. LCER LOAD COMPLEMENT	7-83
	. LDER LOAD SHORT TO LONG	7-97
	. LE, LER LOAD	7-87
	. LNER LOAD NEGATIVE	7-88
	. LPER LOAD POSITIVE	7-95
	. LRER LOAD ROUNDED	7-96
	. LTER LOAD AND TEST	7-80

Type	Description	Pages
Documentation Changes (continued)	. ME, MER MULTIPLY	7-108
	. SE, SER SUBTRACT NORMALIZED	7-165
	. STE STORE	7-157
	Relative addressing version of existing instructions:	
	. RBAL BRANCH AND LINK	7-15
	. RBALS BRANCH AND LINK STACK	7-18
	. RBC BRANCH CONDITIONAL	7-19
	. RBCT BRANCH ON COUNT	7-24
	. RBCX BRANCH CONDITIONAL INDEXED	7-21
	. RBXH BRANCH INDEX HIGH	7-25
	. RBXLE BRANCH INDEX LOW OR EQUAL	7-27
	. RLA LOAD ADDRESS	7-78
	. RPUSHA PUSH ADDRESS	7-125

This manual is updated by: Addendum 800-1100PO-04.01

INDEX

This manual is updated by: Addendum 800-1100PO-04.01

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TELEPHONE (617) 459-5000, TWX 710-343-6769, TELEX 94-7421

WANG

TO:. VS System Users

FROM: Corporate Publications Department

SUBJ: Addendum to the VS Principles of Operation Manual (800-1100PO-04.01)

DATE: September 1982

This addendum chiefly describes VS architecture for systems based on the VS25 and VS100 CPs. Changes have been made to physical and virtual address format, page tables and address translation, the I/O subsystem, and low memory and register assignments. Diagrams of system architecture are included. The addendum also describes the decimal floating-point instructions, and makes some minor corrections to other sections of the manual.

To update your manual, replace existing pages with like-numbered new pages and appropriate point pages; pages DH-1 through DH-3 come just before the index.

Thank you,

Corporate Publications Department

WANG

Customer Comment Form

Title VS PRINCIPLES OF OPERATION

Publications Number 800-1100PO-04.01

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 ______________________
Title ______________________
Dept/Mail Stop ______________________
Company ______________________

Street ______________________
City ______________________
State/Country ______________________
Zip Code __________ **Telephone** ______________________

Thank you for your help.

All comments and suggestions become the property of Wang Laboratories, Inc.

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.
CHARLES T. PEERS, JR., MAIL STOP 1363
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Fold

Cut along dotted line.

WANG

WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL. (617) 459-5000, TWX 710-343-6769, Telex 94-7421

Printed in U.S.A.
800-1100PO-04
8-82-8M