SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS REVISION

Operating System/3 (OS/3)

Assembler (Series 90)
For System 80 see UP-8914

Programmer Reference

UP-8227 Rev. 2

This Library Memo announces the release and availability of "SPERRY UNIVAC® Operating System/3 (OS/3) Assembler Programmer Reference", UP-8227 Rev. 2.

This revision documents the following enhancements to the assembler for release 8.0:

- The display of error messages on the console
- An additional warning message when using continuation characters with macroinstructions

This revision also includes minor technical corrections to material applicable to the assembler prior to release 8.0.

Destruction Notice: If you are going to OS/3 release 8.0, use this revision and destroy all previous copies. If you are not going to OS/3 release 8.0, retain the copy you are now using and store this revision for future use.

Copies of UP-8227 Rev. 1, UP-8227 Rev. 1–A, UP-8227 Rev. 1–B, UP-8227 Rev. 1–C, UP-8227 Rev. 1–D and UP-8227 Rev. 1–E will be available for 6 months after the release of 8.0. Should you need additional copies of these editions, you should order them within 90 days of the release of 8.0. When ordering the previous edition of a manual, be sure to identify the exact revision and update packages desired and indicate that they are needed to support an earlier release.

Additional copies may be ordered by your local Sperry Univac representative.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS:
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A18, 75, and 76 (Cover and 345 pages)	Library Memo for UP-8227 Rev. 2
		RELEASE DATE: September, 1982

UD1-251 Rev. 3/73

Assembler

OS/3

Programmer Reference

Environment: 90/25, 30, 30B, 40 Systems

H

SPERRY UNIVAC

UP-8227 Rev. 2

This document was prepared by Systems Publications using the SPERRY UNIVAC UTS 400 Text Editor. It was printed and distributed by the Customer Information Distribution Center (CIDC), 555 Henderson Rd., King of Prussia, Pa., 19406.

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: UP-8227 Rev. 2
RELEASE LEVEL: 8.0 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		
PSS	1	
Preface	1	
Contents	1 thru 10	
Section 1	Title Page 1 thru 5	
Section 2	Title Page 1 thru 48 48a 49 thru 62 62a, 62b 63 thru 68 68a 69 thru 80 80a 81 thru 120 120a 121 thru 128 128a 129 thru 138 138a thru 138e 139 thru 146 146a 147 thru 163	
Section 3	Title Page 1 thru 31	
Section 4	Title Page 1 thru 29	
Appendix A	Title Page 1 thru 24	
Appendix B	Title Page 1 thru 7	
Appendix C	Title Page 1 thru 13	
Appendix D	Title Page 1 thru 8	
Appendix E	Title Page 1 thru 5	
Appendix F	Title Page 1 thru 6	
Glossary	1 thru 17	
User Comment Sheet		

All the technical changes are denoted by an arrow (→) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (→) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Preface

This programmer reference manual is one in a series designed to be used as a quick-reference document for programmers familiar with the SPERRY UNIVAC Operating System/3 (OS/3). This particular manual describes the basic assembly language (BAL) instructions, directives, and macro definition statements that allow you to write your own assembly language programs and procedure definitions (procs).

No extensive introductory information or examples of use are provided. This type of information is presented in two other assembler manuals: an introduction to the assembler, UP-8030, and an assembler user guide, UP-8061.

The information contained in this manual is presented as follows:

- Section 1. General Information

  Provides a brief overview of the assembler, the job control stream requirements of the assembler, and the conventions that must be observed when reading and writing assembler code.

- Section 2. BAL Application Instructions

  Describes each of the BAL application instructions recognized by the OS/3 assembler. These descriptions are presented in alphabetic order by their operation code mnemonic.

- Section 3. BAL Directives

  Describes each of the directives that are used to control the operation of the assembler. These directives are also presented in alphabetic order by their operation code mnemonic.

- Section 4. BAL Macro Definition Statements

  Describes the macro definition statements used to write and call procedure definitions. These statements are presented in alphabetic order.

- Appendixes

  Contain assembler references, character set code references, math references, source corrections, and system variable symbols helpful to the BAL programmer.

- Glossary

  Defines the terms, expressions, and abbreviations peculiar to the assembler.

Contents

TABLES

1. General Information

ASSEMBLER OVERVIEW

The SPERRY UNIVAC Operating System/3 (OS/3) assembler permits highly-efficient, machine-instruction programs to be written in symbolic form. The assembler consists of an instruction translator and a macro facility. The instruction translator converts symbolic instructions to machine instructions on a one-to-one basis. The macro facility allows a subroutine to be coded, assigned a name, stored in a permanent library, and then to be included in a source program by a simple reference to the subroutine's name in a single instruction. The macro facility greatly reduces the amount of repetitive coding required for routines used frequently within a program or in many different programs.

The assembler accepts source-image input from punched cards, magnetic tape, and disc. It reads source statements and produces a relocatable object module. The object module can then be linked to other object modules to form one load module that is suitable for loading and execution on a SPERRY UNIVAC 90/30 System.

A set of assembler directives is provided to aid you in your program organization and in directing the course of an assembly. All assembly runs produce a printed listing that lists source code, object code, label cross-references, cross-references, and, when necessary, error diagnostics. The final error statement message, which gives the total number of statements flagged in the assembly, is also displayed on the console upon completion of the assembly.

JOB CONTROL REQUIREMENTS

The job control statements required to assemble, linkage edit, and execute are:

LABEL 1	ΔOPERATIONΔ 10	OPERAND 16	Δ	COMMENTS
// JOB	jobname			NAME JOB
// ASMLG				ASSEMBLE, LINK, EXECUTE
/$				START OF DATA
SOURCE	CODE	PROGRAM		
/*				END OF DATA
/&				END OF JOB
// FIN				CLOSE CARD READER

ASSEMBLER CODING FORM

Using an assembler coding form eases the job of writing the program, both for yourself and for the keypunch operator, who must prepare the punched card deck from your written program. Columns 9 and 15 are ruled to remind you that the symbol and operation fields must be terminated by at least one blank.

Symbol Field

The first eight columns of the assembler coding form may contain a symbol. An asterisk (*) indicates that this coding line does not contain instructions and that it contains only comments. The rules for using the symbol field are:

1. The symbol must start in column 1.

2. The symbol must begin with an alphabetic character or special letter.

3. The symbol must not exceed eight characters in length.

4. The symbol must not contain embedded blanks or other special characters.

5. The field must be terminated by a blank.

Operation Field

The operation code is written in the operation field (columns 10 through 14). These codes specify the operation to be performed. The rules for using this field are:

1. The operation code must not contain embedded blanks.

2. The operation code must be written exactly as shown in the list of mnemonics for application instructions, directives, and macro or proc instructions.

3. The operation field must be terminated by a blank.

4. The operation code must not start in column 1.

Operand Field

The operand field begins in column 16 and usually ends in or before column 71. The operands that form part of the assembler statements are written in this field. The rules for using this field are:

1. The operand field is terminated by a blank that is not enclosed by apostrophes.

2. Operands may be continued onto the next line by placing a nonblank character in column 72. Up to two continuation lines are permitted.

3. Continuation lines start in column 16.

Comment Field

Operand specification is usually completed by column 40, thus leaving columns 41 through 71 free for comments. There must be at least one blank between the end of the operand specification and the start of the comments. Long comments can be entered by coding an * in column 1.

Continuation Column

When the operand specification is continued onto the next line, a nonblank character must be written in column 72. Do not confuse this with continuing a comment. An operand specification can be continued for a total of three lines. The second and third continuation lines start in column 16.

Sequence Field

Columns 73 through 80 may be used for entering sequence numbers. This is done by assigning consecutive numbers to each line of coding and is useful for reassembling the card deck if it should be dropped.

READING INSTRUCTION NOTATIONS

Throughout this manual, notations are used to describe the general forms of programmer-written and computer-generated formats. A complete consolidated listing of all the notations is given in A.1.

Assembler Application Instruction Notations

There are six forms of assembler application instructions:

RR — Register-to-register

RX — Register-to-indexed-storage or storage-to-indexed-register

RS — Register-to-nonindexed-storage or storage-to-nonindexed-register

SI — Storage immediate

SS — Storage-to-storage (type SS1)

SS — Storage-to-storage (type SS2)

All of the assembler application instructions and other information are explained in formats that you can write and in the assembler format that generates the machine coding. The following assembler application move instruction (MVC) is an SS1 type:

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	MVC	$d_1(l_1,b_1),d_2(b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	MVC	$s_1(l_1),s_2$

After this application instruction is assembled, it is in the following form:

opcode	l_1	b_1	d_1
0 7	8 15	16 19	20 31

b_2	d_2
32 35	36 47

Table A—1 shows the six formats as generated by the assembler in machine code, as well as the explicit and implicit formats for the programmer coding.

Notation Rules and Meanings

The following conventions are used in application instruction, assembler directive, macro instruction, proc, and control statement formats:

- Optional information is enclosed in brackets [ ] and may be specified or omitted.

  For example:

  [symbol]

- Braces { } indicate multiple options, at least one of which must be chosen.

  For example:

  $$\textbf{PRINT} \left\{ \begin{array}{l} \textbf{ON} \\ \textbf{OFF} \end{array} \right\}$$

- Braces within brackets signify that one of the options must be chosen if that operand is specified.

  For example:

  $$\left[\left\{ \begin{array}{l} \text{DATE} \\ \text{EXT} \\ \text{ID} \\ \text{PRE} \end{array} \right\} \right]$$

- When given a choice of multiple options, the option that is shaded is the default option and indicates the choice that is made by the system if you do not specify one of the options.

  For example:

$$\left[\left\{\begin{array}{l}\text{DATE}\\ \text{EXT}\\ \text{ID}\\ \text{PRE}\end{array}\right\}\right]$$

- Uppercase letters, terms, and punctuation marks indicate information that must be coded exactly as shown.

  For example:

  Mnemonic codes MVN, PACK, and CLC are uppercase.

- Lowercase letters and terms indicate variables that are supplied by you.

  For example:

  [symbol]

- An ellipsis, a series of three periods, indicates that a series of entries may be coded.

  For example:

  $r,[,r_2,\ldots,r_n]$

- Keyword parameters may be coded in any order.

  For example:

  IOROUT=LOAD,BLKSIZE=512,RECFORM=FIXBLK
  BLKSIZE=512,IOROUT=LOAD,RECFORM=FIXBLK

- Positional parameters must be coded in the order shown. Commas are required after each positional parameter except the last. When a positional parameter is omitted from a series of positional parameters, the comma must be retained to indicate the omission.

  For example:

  // JOB Q003,,30,8000,C000
  // JOB Q003,,30,8000

- Throughout this book, the register notations R0 through R15 represent the registers 0 through 15.

  For example:

  BALR   R2,R3

2. BAL Application Instructions

A

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
A	5A	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ■ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the value of operand 2, a full word in main storage, to be algebraically added to operand 1, a general register; the results are placed in operand 1.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	A	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	A	$r_1,s_2(x_2)$

Operational Considerations:

- Operand 2 must be on a full-word boundary address.
- Operand 2 must contain data in fixed-point binary format.
- A fixed-point overflow condition is produced when a value greater than $2^{31}-1$ or -2^{31} is reached in operand 1 (r_1). After overflow, the sign and value of the result are incorrect.
- The contents of operand 2 remain unchanged.

AD*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AD	**6A**	**RX**	**4**

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT <0, SET TO 1
■ IF RESULT >0, SET TO 2
□ IF OVERFLOW, SET TO 3
□ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	■ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
■ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the contents of the double word in storage specified by operand 2 to be algebraically added to the contents of the double-word register specified by operand 1 (r_1). The sum is normalized and placed in the operand 1 (r_1) register.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AD	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AD	$r_1,s_2(x_2)$

* *AD is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

ADR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
ADR	**2A**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ■ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the double-word register specified by operand 2 (r_2) to be algebraically added to the contents of the double-word register specified by operand 1 (r_1). The sum is normalized and placed in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	ADR	r_1,r_2

* *ADR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

AE*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AE	7A	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ■ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the full word in storage specified by operand 2 to be algebraically added to the contents of a full word in the register specified by operand 1 (r_1). The sum is normalized and placed in the full word in the operand 1 (r_1) register.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AE	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AE	$r_1,s_2(x_2)$

* *AE is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

AER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AER	**3A**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ■ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a full word in the register specified by operand 2 (r_2) to be algebraically added to a full word in the register specified by operand 1 (r_1). The sum is normalized and placed in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AER	r_1,r_2

* *AER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

AH

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AH	4A	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ■ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the value of operand 2, a half word in main storage, to be algebraically added to operand 1, a general register; the results are placed in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	AH	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	AH	$r_1,s_2(x_2)$

Operational Considerations:

- Operand 2 must be on a half-word boundary address.
- Operand 2 must contain data in fixed-point binary format.
- A fixed-point overflow condition is produced when a value greater than $2^{31}-1$ or -2^{31} is reached in operand 1 (r_1). After overflow, the sign and value of the result are incorrect.
- The contents of operand 2 remain unchanged.

AI

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AI	9A	SI	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ■ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ■ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the value of operand 2, immediate data, to be algebraically added to operand 1, a half word in main storage; the results are placed in operand 1.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AI	$d_1(b_1),i_2$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AI	s_1,i_2

Operational Considerations:

- Operand 1 must be on a half-word boundary address.
- Operand 1 must contain data in fixed-point binary format.
- A fixed-point overflow condition is produced when a value greater than $2^{15}-1$ or -2^{15} is reached in operand 1. After overflow, the sign and value of the result are incorrect.
- The maximum value for operand 2 (i_2) is +127 or —128.

AL*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AL	5E	RX	4

Condition Codes
■ SET TO 0
■ SET TO 1
■ SET TO 2
■ SET TO 3
SEE OPER. CONSIDERATIONS

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of operand 2, a full word in storage, to be logically added to the contents of the full word in the operand 1 (r_1) register. The sum is placed in operand 1 (r_1).

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AL	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AL	$r_1,s_2(x_2)$

Operational Considerations:

- Logical addition is performed by adding all 32 bits of each operand.
- The contents of operand 2 remain unchanged.
- Operand 2 must be a full word, in storage, on a full-word boundary.

* *AL is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

- The condition code is set:
  - to zero if result is zero; no carryout of most significant bit;
  - to 1 if result is not zero; no carryout of most significant bit;
  - to 2 if result is zero; carryout of most significant bit; or
  - to 3 if result is not zero; carryout of most significant bit.

ALR*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
ALR	1E	RR	2

Condition Codes
■ SET TO 0
■ SET TO 1
■ SET TO 2
■ SET TO 3
SEE OPER. CONSIDERATIONS

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of the operand 1 (r_1) and operand 2 (r_2) registers to be logically added. The sum is placed in operand 1 (r_1).

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	ALR	r_1,r_2

Operational Considerations:

- Logical addition is performed by adding all 32 bits of each operand.
- The contents of operand 2 (r_2) remain unchanged.
- The condition code is set:
  - to zero if result is zero; no carryout of most significant bit;
  - to 1 if result is not zero; no carryout of most significant bit;
  - to 2 if result is zero; carryout of most significant bit; or
  - to 3 if result is not zero; carryout of most significant bit.

* *ALR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

AP

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AP	FA	SS	6

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT < 0, SET TO 1
■ IF RESULT > 0, SET TO 2
■ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
■ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
■ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Algebraically adds the contents of operand 2 (a packed number in main storage) to operand 1 (also a packed number in main storage). The result is stored in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	AP	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	AP	$s_1(l_1),s_2(l_2)$

Operational Considerations:

- All signs and digits are checked for validity and the sign of the result is determined algebraically.
- A zero result has a positive sign when the operation is completed without overflow.
- Operand 1 and operand 2 must be packed numbers.
- When most significant digits are lost because of overflow, the partial result has the sign that the correct result would have had.

AP

- If operand 2 is shorter than operand 1, operand 2 is extended with zero digits.
- An overflow condition results if the capacity of the operand 1 field is exceeded by the result or if the carryout of the most significant digit position of the result field is lost.
- Operand 1 and operand 2 may overlap if their least significant bytes coincide. This makes it possible to add a number to itself.

AR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AR	1A	RR	2

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ■ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the value of operand 2 (r_2) to be algebraically added to the value of operand 1 (r_1). The results are placed in operand 1.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	AR	r_1,r_2

Operational Considerations:

- A fixed-point overflow condition is produced when a value greater than $2^{31}-1$ or -2^{31} is reached in operand 1. After overflow, the sign and value of the result are incorrect.
- The contents of the register for operand 2 (r_2) remain unchanged.

AU*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AU	7E	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- ■ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the contents of the full word in storage specified by operand 2 to be algebraically added to the contents of a full word in the register specified by operand 1 (r_1). The sum is placed in the operand 1 (r_1) register.

Explicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	AU	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	AU	$r_1,s_2(x_2)$

Operational Consideration:

- The execution of the AU instruction is identical to that of the AE instruction, except that the sum is not normalized before being placed in operand 1.

* *AU is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

AUR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AUR	**3E**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	■ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of a full word in the register specified by operand 2 (r_2) to be algebraically added to a full word in the register specified by operand 1 (r_1). The sum is placed in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AUR	r_1,r_2

Operational Consideration:

- The execution of the AUR instruction is identical to that of the AER instruction, except that the sum is not normalized before being placed in operand 1.

* *AUR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

AW*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AW	**6E**	**RX**	**4**

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT < 0, SET TO 1
■ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	■ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of a double word in storage specified by operand 2 to be algebraically added to the contents of the double word in the register specified by operand 1 (r_1). The sum is placed in the double word in the register specified by operand 1 (r_1).

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AW	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	AW	$r_1,s_2(x_2)$

Operational Consideration:

- The execution of the AW instruction is identical to that of the AD instruction, except that the sum is not normalized before being placed in operand 1 (r_1).

* *AW is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

AWR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
AWR	2E	RR	2

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- ■ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the contents of the double-word register specified by operand 2 (r_2) to be algebraically added to the double-word contents of operand 1 (r_1). The sum is placed in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	AWR	r_1,r_2

Operational Consideration:

- The execution of the AWR instruction is identical to that of the ADR instruction, except that the sum is not normalized before being placed in operand 1 (r_1).

* *AWR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

BAL

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BAL	45	RX	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Loads the address of the next sequential instruction into the register in the first operand and then branches to the location specified in the second operand. The normal sequence of instructions may be reinstated when a return branch via r_1 is taken. BAL is an unconditional branch instruction.

NOTE:

Bits 32 through 39 (instruction length code, condition code, and program mask) of the current program status word (PSW) are stored in bit positions 0 through 7 of operand 1 (r_1). The return address is stored in bits 8 through 31 of operand 1 (r_1).

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	BAL	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	BAL	$r_1,s_2(x_2)$

BALR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BALR	05	RR	2

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	■ NONE

Function:

Loads the relative address of the next sequential instruction into the first operand register and then branches to the address in the second operand register. The normal sequence of instructions may be reinstated when a return branch via r_1 is taken. When the second operand (r_2) is zero, there is no branch and the next sequential instruction is executed.

NOTE:

Bits 32 through 39 (instruction length code, condition code, and program mask) of the current program status word (PSW) are stored in bit positions 0 through 7 of operand 1 (r_1). The return address is stored in bits 8 through 31 of operand 1 (r_1).

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	BALR	r_1,r_2

BAS
BASR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BAS & BASR	**4D & 0D**	**RX & RR**	**4 or 2**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

These instructions do not exist in the native mode instruction set and are used only when operating in the 360/20 compatibility mode.

BC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BC	47	RX	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	■ NONE

Function:

Checks the specified mask (m_1), operand 1, with the current condition code. If any 1 bits match, a branch takes place to the location specified by operand 2; otherwise, the next sequential instruction is executed. See Table A—3 for the list of BC formats and equivalent extended mnemonic codes.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	BC	$m_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	BC	$m_1,s_2(x_2)$

Operational Considerations:

- The mask, operand 1, determines the condition code setting in the PSW to be tested, as follows:
  - An 8 produces the mask 1000_2, which tests bit 8 for a zero condition code.
  - A 4 produces the mask 0100_2, which tests bit 9 for a 1 condition code.

BC

- A 2 produces the mask 0010_2, which tests bit 10 for a 2 condition code.
- A 1 produces the mask 0001_2, which tests bit 11 for a 3 condition code.
- A zero produces the mask 0000_2, which is equivalent to no-operation.
- Any combination of 1's and zeros in the mask tests for more than one condition code.
- Any 1 bit on and tested produces the branch.

- A mask specification of 15 (1111_2) produces an unconditional branch.

BCR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BCR	07	RR	2

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Checks the specified mask (m_1), operand 1, with the current condition code. If any 1 bits match, a branch takes place to the location specified by operand 2 (r_2); otherwise, the next sequential instruction is executed. If operand 2 (r_2) is zero, no branch will take place. See Table A—3 for the list of BC formats and equivalent extended mnemonic codes.

Implicit and Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	BCR	m_1,r_2

Operational Considerations:

- The mask, operand 1, determines the condition code setting in the PSW to be tested, as follows:
  - — An 8 produces the mask 1000_2, which tests bit 8 for a zero condition code.
  - — A 4 produces the mask 0100_2, which tests bit 9 for a 1 condition code.
  - — A 2 produces the mask 0010_2, which tests bit 10 for a 2 condition code.
  - — A 1 produces the mask 0001_2, which tests bit 11 for a 3 condition code.
  - — A zero produces the mask 0000_2, which is equivalent to no-operation.
  - — Any combination of 1's and zeros in the mask tests for more than one condition code.
  - — Any 1 bit on and tested produces the branch.
- A mask specification of 15 (1111_2) produces an unconditional branch.

BCT

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BCT	46	RX	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	■ NONE

Function:

Each time this instruction is executed, the value in r_1 is decremented by 1 and then tested to see whether the result is equal to zero. If the result is not equal to zero, a branch takes place to the location specified by operand 2. If the result is equal to zero, then no branch takes place and the next sequential instruction is executed. This instruction can be used to control the number of times a loop routine is executed. The initial value in r_1 must be a positive value greater than zero.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	BCT	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	BCT	$r_1,s_2(x_2)$

BCTR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BCTR	06	RR	2

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

BCTR is the RR format type of BCT and works in the same way, except the second operand (r_2) is a register rather than a storage location. The BCTR instruction is initiated by loading a value in the first operand register (r_1) to be used as a count value and a branch address into the second operand register (r_2). Each time this instruction is executed, the value in r_1 is decremented by 1 and then tested to see whether the result is equal to zero. If the result is not equal to zero, a branch takes place to the address in the second operand (r_2). If the result is equal to zero, then no branch takes place and the next sequential instruction is executed. This instruction can be used to control the number of times a loop routine is executed. The initial value in r_1 must be a positive value greater than zero. If the second operand (r_2) is zero, no branch will take place.

Implicit and Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	BCTR	r_1,r_2

BXH*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BXH	86	RS	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT $<$0, SET TO 1
☐ IF RESULT $>$0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Compares the algebraic sum of operand 1 (r_1) and operand 2 (r_3) to a value that is equal to the number of the register specified as operand 2 (r_3) or $r_3 + 1$. If the sum of operand 1 (r_1) and operand 2 (r_3) is less than or equal to the compare value, the next sequential instruction is executed; if the sum is greater than the compare value, then a branch will take place to the location specified by operand 2, which is d_2 (b_2) or s_2. The value being used as the reference is always an odd-numbered register and is specified by r_3 if r_3 is an odd-numbered register, or is $r_3 +1$ if r_3 is an even-numbered register. Following the comparison, the sum is placed in the first operand location. All quantities are treated as signed integers. An operation exception takes place if this operation is attempted on a processor that does not have this feature installed.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	BXH	$r_1,r_3,d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	BXH	r_1,r_3,s_2

* *BXH is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

BXLE*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
BXLE	**87**	**RS**	**4**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

This instruction is the same as BXH, except that the branch is made when the sum of the first operand (r_1) and the third operand (r_3) is less than or equal to the value being compared.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	BXLE	$r_1,r_3,d_2(b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	BXLE	r_1,r_3,s_2

* *BXLE is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

C

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
C	59	RX	4

Condition Codes

- ■ IF r_1 = OPERAND 2, SET TO 0
- ■ IF r_1 < OPERAND 2, SET TO 1
- ■ IF r_1 > OPERAND 2, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of operand 1 (r_1) to be algebraically compared with the contents of operand 2, a full word in main storage.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	C	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	C	$r_1,s_2(x_2)$

Operational Considerations:

- The contents of both operands remain unchanged.
- Operand 2 must be on a full-word boundary.

CD*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CD	**69**	**RX**	**4**

Condition Codes

■ IF OP1 = OP2, SET TO 0
■ IF OP1 < OP2, SET TO 1
■ IF OP1 > OP2, SET TO 2
□ IF OVERFLOW, SET TO 3
□ UNCHANGED

Possible Program Exceptions

■ ADDRESSING
□ DATA (INVALID SIGN/DIGIT)
□ DECIMAL DIVIDE
□ DECIMAL OVERFLOW
□ EXECUTE
□ EXPONENT OVERFLOW
□ EXPONENT UNDERFLOW
□ FIXED-POINT DIVIDE
□ FIXED-POINT OVERFLOW
□ FLOATING-POINT DIVIDE
■ OPERATION
■ PROTECTION
□ SIGNIFICANCE
■ SPECIFICATION:
■ NOT A FLOATING-POINT REGISTER
□ OP 1 NOT ON HALF-WORD BOUNDARY
□ OP 2 NOT ON HALF-WORD BOUNDARY
□ OP 2 NOT ON FULL-WORD BOUNDARY
■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ OP 1 NOT EVEN NUMBERED REGISTER
□ OP 1 NOT ODD NUMBERED REGISTER
□ NONE

Function:

Causes the contents of a double word in the register specified by operand 1 (r_1) to be algebraically compared with the contents of a double word in storage specified by operand 2. The condition code is set by this instruction.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CD	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CD	$r_1,s_2(x_2)$

Operational Considerations:

- Comparison is accomplished by the rules for normalized fixed-point subtraction. The operands are equal when the intermediate sum, including the guard digit, is zero.

- Operands with zero fractions compare as equal even when their signs or exponents are different.

* *CD is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

CDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CDR	29	RR	2

Condition Codes

- ■ IF OP1 = OP2, SET TO 0
- ■ IF OP1 < OP2, SET TO 1
- ■ IF OP1 > OP2, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of a double word in the register specified by operand 1 (r_1) to be algebraically compared with the contents of a double word in the register specified by operand 2 (r_2). The condition code is set by this instruction.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CDR	r_1,r_2

Operational Considerations:

- Comparison is accomplished by the rules for normalized fixed-point subtraction. The operands are equal when the intermediate sum, including the guard digit, is zero.
- Operands with zero fractions compare as equal even when their signs or exponents are different.

* *CDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

CE*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CE	79	RX	4

Condition Codes

- ■ IF OP1 = OP2, SET TO 0
- ■ IF OP1 < OP2, SET TO 1
- ■ IF OP1 > OP2, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the contents of a full word in the register specified by operand 1 (r_1) to be algebraically compared with the contents of a full word word in storage specified by operand 2. The condition code is set by this instruction.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CE	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CE	$r_1,s_2(x_2)$

Operational Considerations:

- Comparison is accomplished by the rules for normalized fixed-point subtraction. The operands are equal when the intermediate sum, including the guard digit, is zero.
- Operands with zero fractions compare as equal even when their signs or exponents are different.

* *CE is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

CER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CER	39	RR	2

Condition Codes

- ■ IF OP1 = OP2, SET TO 0
- ■ IF OP1 < OP2, SET TO 1
- ■ IF OP1 > OP2, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the full-word contents of the register specified by operand 1 (r_1) to be algebraically compared with the contents of a full word in the register specified by operand 2 (r_2). The condition code is set by this instruction.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CER	r_1,r_2

Operational Considerations:

- Comparison is accomplished by the rules for normalized fixed-point subtraction. The operands are equal when the intermediate sum, including the guard digit, is zero.

- Operands with zero fractions compare as equal even when their signs or exponents are different.

* *CER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

CH

General				Possible Program Exceptions	
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)	■ ADDRESSING	■ PROTECTION
MNEM.	HEX.			□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
CH	49	RX	4	□ DECIMAL DIVIDE	■ SPECIFICATION:
				□ DECIMAL OVERFLOW	□ NOT A FLOATING-POINT REGISTER
				□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
				□ EXPONENT OVERFLOW	■ OP 2 NOT ON HALF-WORD BOUNDARY
				□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
				□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
				□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
				□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
				□ OPERATION	□ NONE

Condition Codes
■ IF r_1 = OPERAND 2, SET TO 0
■ IF r_1 < OPERAND 2, SET TO 1
■ IF r_1 > OPERAND 2, SET TO 2
□ IF OVERFLOW, SET TO 3
□ UNCHANGED

Function:

Causes the contents of operand 1 (r_1) to be algebraically compared with the contents of operand 2 (a half word in main storage), after operand 2 is expanded, by propagating the sign bit to fill a full word.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CH	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CH	$r_1,s_2(x_2)$

Operational Considerations:

- The contents of both operands remain unchanged.
- Operand 2 must be on a half-word boundary.

CL

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CL	55	RX	4

Condition Codes

- ■ IF r_1 = OPERAND 2, SET TO 0
- ■ IF r_1 < OPERAND 2, SET TO 1
- ■ IF r_1 > OPERAND 2, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a full word in storage specified by operand 2 to be compared with the contents of the register specified by operand 1 (r_1). The condition code is set according to the comparison result.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CL	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CL	$r_1,s_2(x_2)$

Operational Considerations:

- Operands are considered unsigned binary numbers and all bit combinations are valid.
- The contents of both operands remain unchanged.
- Operand 2 must be on a full-word boundary.

CLC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CLC	D5	SS	6

Condition Codes
■ IF OP1 = OP2, SET TO 0
■ IF OP1 < OP2, SET TO 1
■ IF OP1 > OP2, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes the contents of one area in main storage specified by operand 1 to be compared with an equal length area in main storage specified by operand 2. The condition code is set according to the comparison result.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CLC	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CLC	$s_1(l),s_2$

Operational Considerations:

- The l specification of operand 1 specifies the length of both operands.
- Operands are considered unsigned binary numbers and all bit combinations are valid.
- The contents of both operands remain unchanged.
- The instruction is processed from left to right, byte by byte.
- If the number of bytes to be compared is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

CLI

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CLI	95	SI	4

Condition Codes

- ■ IF OPERAND 1 = i_2, SET TO 0
- ■ IF OPERAND 1 < i_2, SET TO 1
- ■ IF OPERAND 1 > i_2, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of one byte in main storage specified by operand 1 to be compared with the one byte of immediate data specified in operand 2. The condition code is set according to the comparison result.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CLI	$d_1(b_1),i_2$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CLI	s_1,i_2

Operational Considerations:

- Operands are considered unsigned binary numbers and all bit combinations are valid.
- Operands are one byte in length.
- The contents of operand 1 remain unchanged.

CLR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CLR	15	RR	2

Condition Codes

- ■ IF $r_1 = r_2$, SET TO 0
- ■ IF $r_1 < r_2$, SET TO 1
- ■ IF $r_1 > r_2$, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Causes the contents of the operand 1 (r_1) register to be compared with the contents of the operand 2 (r_2) register. The condition code is set according to the comparison result.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CLR	r_1,r_2

Operational Considerations:

- Operands are considered unsigned binary numbers and all bit combinations are valid.
- The contents of both operands remain unchanged.

CP

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CP	F9	SS	6

Condition Codes

- ■ IF OP1 = OP2, SET TO 0
- ■ IF OP1 < OP2, SET TO 1
- ■ IF OP1 > OP2, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ■ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- □ OPERATION
- ■ PROTECTION
- □ SIGNIFICANCE
- □ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Compares the contents of two storage areas to see whether they are algebraically equal, operand 1 is higher, or operand 1 is lower. The condition code is set to reflect the results of this compare. A branch instruction is usually used after the compare instruction.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CP	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CP	$s_1(l_1),s_2(l_2)$

Operational Considerations:

- All signs and digits are checked for validity, and comparison proceeds from right to left.
- If the operand fields are unequal in length, the shorter field is extended with zero digits.
- Operands with zero values and unlike signs compare as equal.
- All valid codes representing the same sign are considered equal.
- Operand 1 and operand 2 may overlap if their least significant bytes coincide.
- The contents of both operands remain unchanged.

CR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CR	19	RR	2

Condition Codes

- ■ IF $r_1 = r_2$, SET TO 0
- ■ IF $r_1 < r_2$, SET TO 1
- ■ IF $r_1 > r_2$, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Causes the contents of operand 1 (r_1) to be algebraically compared to operand 2 (r_2).

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CR	r_1,r_2

Operational Consideration:

- The contents of both registers remain unchanged.

CVB

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CVB	4F	RX	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ■ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ■ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Converts the packed decimal number in operand 2, a double word in main storage, to a fixed-point signed binary number, which is placed in operand 1 (r_1).

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CVB	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CVB	$r_1,s_2(x_2)$

Operational Considerations:

- Operand 2 is a 15-digit and sign packed decimal number in a double word on a double-word boundary in main storage.

- Operand 2 is checked for valid digits and sign code before conversion to a fixed-point, 32-bit signed binary number.

CVB

- The maximum number that can be converted and still contained in a 32-bit register is 2,147,483,647 ($2^{31}-1$). The minimum number is —2,147,483,648 (-2^{31}). For decimal numbers exceeding this range, the 32 least significant bits are stored in the first operand location and a fixed-point divide exception is generated.

- If operand 2 is negative, the result will be in twos complement notation.

- The contents of operand 2 remain unchanged.

CVD

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
CVD	4E	RX	4

Condition Codes

- □ IF RESULT = 0, SET TO 0
- □ IF RESULT < 0, SET TO 1
- □ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	□ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
□ OPERATION	□ NONE

Function:

Converts the fixed-point signed binary number in operand 1 (r_1) to a packed decimal number, which is placed in operand 2, a double word in main storage.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CVD	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CVD	$r_1,s_2(x_2)$

Operational Considerations:

- Operand 1 is a fixed-point, 32-bit signed binary number in a register.
- Operand 2 is a 15-digit packed signed decimal number in a double-word main storage location on a double-word boundary.
- The contents of operand 1 remain unchanged.

D

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
D	5D	RX	4

Condition Codes

- □ IF RESULT = 0, SET TO 0
- □ IF RESULT < 0, SET TO 1
- □ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- ■ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- □ OPERATION
- ■ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ■ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the value in the even-odd pair of registers specified by operand 1 (r_1) to be divided by the full-word operand 2 (the divisor). The quotient and remainder are placed in the operand 1 registers.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	D	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	D	$r_1,s_2(x_2)$

Operational Considerations:

- Operand 1 is treated as a 64-bit fixed-point signed binary integer and occupies an even-odd register pair. The operand 1 field of the instruction must specify an even-numbered register. The 32-bit remainder and 32-bit quotient replace the dividend in the even-numbered and odd-numbered register, respectively.
- Operand 2 is treated as a 32-bit fixed-point signed binary integer. The contents of operand 2 remain unchanged after execution.
- The sign of the quotient is determined algebraically, and the remainder assumes the sign of the dividend. A zero quotient or zero remainder is always positive.
- When the quotient exceeds 32 bits or the divisor is equal to zero, a fixed-point divide exception occurs, no division takes place, and the dividend remains unchanged.

DD*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
DD	6D	RX	4

Condition Codes
□ IF RESULT = 0, SET TO 0
□ IF RESULT <0, SET TO 1
□ IF RESULT >0, SET TO 2
□ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
■ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
■ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the double-word contents of the operand 1 (r_1) register to be divided by the contents of the double word in storage specified by operand 2. The normalized quotient is placed in the register specified by operand 1 (r_1). Any remainder is not preserved.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DD	$r_1, d_2(x_2, b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DD	$r_1, s_2(x_2)$

* *DD is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

DDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
DDR	**2D**	**RR**	**2**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
■ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
■ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the double-word contents of the operand 1 (r_1) register to be divided by the double-word contents of the operand 2 (r_2) register. The normalized quotient is placed in the operand 1 (r_1) register. Any remainder is not preserved.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	DDR	r_1,r_2

* *DDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

DE*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
DE	7D	RX	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
■ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
■ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the full-word contents of the operand 1 (r_1) register to be divided by the full-word contents of a full word in storage specified by operand 2. The normalized quotient is placed in a full word in the operand 1 (r_1) register. Any remainder is not preserved.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DE	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DE	$r_1,s_2(x_2)$

* *DE is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

DER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
DER	**3D**	**RR**	**2**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ■ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ■ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the full-word contents of the operand 1 (r_1) register to be divided by the full-word contents of the operand 2 (r_2) register. The normalized quotient is placed in a full word in the operand 1 (r_1) register. Any remainder is not preserved.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DER	r_1,r_2

* *DER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

DIAG

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
DIAG	83	SI	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- □ IF RESULT < 0, SET TO 1
- □ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Resets the processor to zero after control storage is loaded and provides various diagnostic and supervisor operations.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DIAG	$d_1(b_1), i_2$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DIAG	s_1, i_2

DP

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
DP	FD	SS	6

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
■ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
■ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes the contents of operand 1 (the dividend) to be divided by the contents of operand 2 (the divisor). The quotient and remainder are placed in the operand 1 location.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DP	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DP	$s_1(l_1),s_2(l_2)$

Operational Considerations:

- All signs and digits are checked for validity.
- The quotient and remainder occupy the entire operand 1 field. The remainder is right-justified in the field, carries the sign of operand 1, and is equal in size to operand 2. The quotient, carrying the algebraically determined sign, is right-justified in the rest of the field.
- The maximum dividend (operand 1) size is 31 digits and sign. The maximum quotient size is 29 digits and sign. The smallest remainder is one digit and sign. The maximum divisor is 15 digits.

- Operand 1 and operand 2 fields may overlap if their least significant bytes coincide.
- If the number of quotient digits exceeds the size of the quotient field or if division by zero is attempted, a decimal divide exception results; the divisor and dividend remain unchanged in their storage locations.
- A decimal divide exception occurs if the dividend does not have at least one leading zero. The condition for a decimal divide exception can be determined by aligning the leftmost digit of the divisor (operand 2) field with the leftmost less 1 digit of the dividend (operand 1) field and performing a subtraction. If, after alignment, the divisor is less than or equal to the dividend, a decimal divide exception is indicated.
- A specification exception indicates the divisor exceeds 15 digits or operand 1 is not longer than operand 2.

DR*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
DR	1D	RR	2

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ■ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ■ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the value in the even-odd registers specified by operand 1 (r_1) to be divided by the value in the register (the divisor) specified by operand 2 (r_2). The quotient and remainder are placed in the operand 1 registers.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	DR	r_1,r_2

Operational Considerations:

- Operand 1 is treated as a 64-bit fixed-point signed binary integer and occupies an even-odd register pair. The operand 1 field of the instruction must specify an even-numbered register. The 32-bit remainder and 32-bit quotient replace the dividend in the even-numbered and odd-numbered register, respectively.

- Operand 2 is treated as a 32-bit fixed-point signed binary integer. The contents of operand 2 remain unchanged after execution.

- The sign of the quotient is determined algebraically and the remainder assumes the sign of the dividend. A zero quotient or zero remainder is always positive.

- When the quotient exceeds 32 bits or the divisor is equal to zero, a fixed-point divide exception occurs, no division takes place, and the dividend remains unchanged.

- A specification exception will occur if r_1 specifies an odd-numbered register.

* *DR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

ED

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
ED	DE	SS	6

Condition Codes

- ■ SET TO 0
- ■ SET TO 1
- ■ SET TO 2
- ☐ SET TO 3

SEE OPER. CONSIDERATIONS

Possible Program Exceptions

- ■ ADDRESSING
- ■ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the packed data specified by operand 2 to be unpacked and edited under the control of a mask (pattern) specified by operand 1. The result is placed in the main storage location specified by operand 1. This instruction can produce the following types of results:

- Zero suppression
  Ex: 00173 — 173
- Character protection
  Ex: 000453 — ***4.53
- Punctuation
  Ex: 123400 — $1,234.00
- Multiple field editing
  Ex: 12531468 — 12.53△△14.68

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	ED	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	ED	$s_1(l),s_2$

ED

Operational Considerations:

- For every digit in the source field, operand 2, there must be an equal number of digit select characters, significance start characters, or a combination of both in the pattern.

- The significance indicator, referred to as the S switch, indicates by its on or off state the significance or nonsignificance, respectively, of subsequent operand 2 digits or message characters. Significant operand 2 digits replace their corresponding digit select or significance start characters in the result. Significant message characters remain unchanged in the result.

- The S switch is turned off when the *edit* instruction starts and when a sign code of "C" (+) is reached; and it is turned on when the first signficant (nonzero) digit is reached.

- When the S switch is off, zeros to be transferred from operand 2 are suppressed and the fill character is inserted in the corresponding operand 1 position. When the S switch is on, any zero to be transferred from operand 2 is unpacked into the corresponding operand 1 position. At the beginning of execution, the S switch is off.

- Editing includes sign and punctuation control and the suppression and protection of leading zeros. It also facilitates programmed blanking for all zero fields. Several fields may be edited in one operation, and numeric information may be combined with text.

- The instruction proceeds from left to right.

- Operand 2 data must be in packed format and must contain valid numerics and sign codes.

- The original contents of operand 1 is the mask, the pattern which controls the edit process. Depending on the edit requirements, some or most of the bytes originally in operand 1 are replaced by data from operand 2. The mask is expressed in unpacked format and may consist of any combination of 8-bit characters.

- As the mask is scanned from left to right, one of three things happens to each mask character:

  — An operand 2 digit is expanded to a zoned character. The zoned character replaces the mask character. When the operand 2 digit is stored as the result, its code is expanded from packed to unpacked format by attaching a generated zone code.

  — The mask character is left unchanged.

  — A fill character is stored in the result. The fill character is taken from the first byte position of the mask. The choice of this character is not dependent upon the editing function initiated by this code. The editing function occurs after the code has been assigned as a fill character.

ED

Each mask character is replaced by a result character that depends on three conditions:

— the digit obtained from operand 2;

— the mask character; and

— the S switch status.

When a digit select or significance start byte is found in the mask, the S switch and an operand 2 digit are examined. This results in either the unpacked operand 2 digit or the fill character replacing the mask character. A valid decimal digit (if the mask byte is a significance start) or nonzero decimal digit (if the mask byte is a digit select) sets the S switch to on if the operand 2 byte does not contain a plus code in the four least significant bit positions.

- The fill character is the leftmost character of the edit mask (operand 1). Any valid hexadecimal value (B.2) may be used as a fill character. This character is retained for the editing which follows. This position does not receive a digit from the operand 2 data.

- The digit select byte is a character in the operand 1 mask represented by EBCDIC code 20. If the digit select byte is encountered and the S switch is on, any digit, 0 through 9, is unpacked to replace the digit select byte. If the S switch is off, the operand 2 digit is examined and only nonzero digits are unpacked into operand 1. The fill character replaces the digit select byte if the examined digit is zero. The S switch is turned on when the first nonzero operand 2 digit is encountered; this allows succeeding zeros from operand 2 to be included in the result.

- The significance start byte is represented in the edit mask by EBCDIC code 21. The significance start byte performs the same function as the digit select byte except the significance start byte turns the S switch on, regardless of the value of the current operand 2 digit. Once the S switch is on, it remains on for all succeeding digits; however, the current digit is not affected. The S switch may be turned off by a field separator byte or by a positive sign code within operand 2.

- Any other symbol or data in the operand 1 edit mask, as represented by hexadecimal codes, is retained unchanged if the S switch is on. If the S switch is off, this other data is replaced by the fill character. During this operation, the digit of operand 2 is neither accessed nor addressed-advanced.

- The sign of operand 2, positive or negative, must be a value greater than binary 9 (1002_2). Any hexadecimal value A through F is acceptable. The sign itself is not moved to operand 1; instead, a sign indicator, such as a minus sign or letters CR, is either deleted from or retained in operand 1, depending on the sign of operand 2.

  The sign of operand 2 also affects the S switch. A positive sign turns the S switch off, thus causing the following characters in operand 1 to be replaced by the fill character. A negative sign leaves the S switch unchanged.

ED

- If the fill character is a blank, if no significance start byte appears in the mask, and if operand 2 is all zeros, the editing operation blanks the result field.

- Overlapping operand 1 and operand 2 fields produces unpredictable results.

- The length specification (l) in the object instruction specifies the length of the mask (operand 1). The length of the mask can be determined as:

  — one byte for the fill character;

  — one byte for each digit select byte, significance start byte, and field separator byte; and

  — one byte for each message character.

  Usually, operand 2 is shorter than operand 1 because a zone (a half byte) and a numeric (a full byte) are inserted in the result for each operand 2 digit. The total number of digit-select and significance start bytes in the mask must equal the number of operand 2 digits to be edited.

- If operand 2 containing unpacked data is to be edited, it must first be packed by the PACK instruction. In packing an odd number of bytes, an odd number of digit positions and the sign are produced. In packing an even number of bytes, an odd number of digit positions and the sign are produced. The extra digit position in the latter case is zero and is the most significant position in operand 2. The extra position must be provided for in the mask by specifying an extra DSB or SSB. Space, asterisk, or other character fill occurs and may be dropped when transferring the edited operand to output.

- Multiple-field editing operations are indicated by the presence of one or more field separator bytes (EBCDIC code 22). The field separator byte identifies the individual fields in this operation and is always replaced in the mask with a fill character. The S switch is always off after the field separator byte is encountered. If field separators are not indicated by the mask, the entire operand 2 is considered one field.

- The condition code, reflecting the status of the last source field edited, is set:

  — to zero when all of the operand 2 digits in the last field are zero; if the mask of the last field has no significance start or digit select bytes, the operand 2 digits are not examined and the condition code is set to zero;

  — to 1 when a nonzero operand 2 digit is detected and the S switch is set after the last mask digit is examined; or

  — to 2 when a nonzero operand 2 digit is detected and the S switch is off after the last mask digit is examined.

  Code 3 is not used.

ED

- The operation of the *edit* instruction is summarized in the following table.

Mask (Operand 1) Character	EBCDIC Code	S Switch Status	Data (Operand 2) Character	Resulting (Operand 1) Character	Resulting S Switch Status
Fill character	Any	Off	Not examined	None	Off
Digit select byte	20	On	Digit	Digit	On*
		Off	Nonzero	Digit	On*
		Off	Zero	Fill character	Off
Significance start byte	21	On	Digit	Digit	On*
		Off	Nonzero	Digit	On*
		Off	Zero	Fill character	On*
Message character	Any except 20, 21, 22	On	Not examined	Message character	On*
		Off	Not examined	Fill character	Off
Field separator byte	22	On	Not examined	Fill character	Off
		Off	Not examined	Fill character	Off

*Sign detection (examined simultaneously with operand 2 digit) affects the S switch as follows:

1. A plus or minus sign detected as a most significant digit causes a data exception.
2. A plus sign detected as a least significant digit causes the S switch to be turned off.
3. A minus sign has no effect on the S switch.

- If the number of bytes to be edited is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

EDMK*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
EDMK	**DF**	**SS**	**6**

Condition Codes

- ■ SET TO 0
- ■ SET TO 1
- ■ SET TO 2
- ☐ SET TO 3

SEE OPER. CONSIDERATIONS

Possible Program Exceptions

■ ADDRESSING	■ PROTECTION
■ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

This instruction is identical to the *edit* (ED) instruction, except for the additional function of placing the address of the first significant result digit in register 1. This is done to permit the use of a floating $ character or other character in the result field.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	EDMK	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	EDMK	$s_1(l),s_2$

Operational Considerations:

- The *edit and mark* (EDMK) instruction is identical to the *edit* (ED) instruction, except that EDMK inserts the resulting address of the first significant character in the low-order 24 bits of general register 1. This insertion occurs whenever the result character is a zoned source digit and the significant switch is zero before examination of the digit.

- The condition code is set in the same manner as the *edit* instruction.

* *EDMK is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

EDMK

- The *edit and mark* instruction facilitates the programming of floating currency-symbol insertion. The character address inserted in general register 1 is one more than the address where a floating currency sign would be inserted. The *branch on count* (BCTR) instruction, with zero 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 into the register beforehand the address of the pattern character that immediately follows the significance starter.
- When a single instruction is used to edit several fields, the address of the first significant result character of each field is inserted into bit positions 8 through 31 of general register 1. Only the address of the first significant character of the last field is available after the instruction is completed.
- If the number of bytes to be edited is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

EX

General

OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
EX	**44**	**RX**	**4**

Condition Codes

- □ IF RESULT = 0, SET TO 0
- □ IF RESULT <0, SET TO 1
- □ IF RESULT >0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- ■ SEE OPER. CONSIDERATIONS

Possible Program Exceptions

- ■ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- ■ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- □ OPERATION
- ■ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Used to branch to a nonsequential instruction, then to execute it, with or without modification, and then to return to the normal sequence of instructions.

If operand 1 is 0, the instruction at the operand 2 address, specified by d_2 (x_2, b_2), is executed without modification. If operand 1 (r_1) is in the range 1—15, the contents of r_1 are used to modify the subject instruction when that instruction is staticized.

When r_1 is nonzero, modification of the operand 2 instruction proceeds as follows: A logical addition (OR) is performed on the contents of bits 24 through 31 of r_1 and bits 8 through 15 of the operand 2 instruction. The result replaces bits 8 through 15 of the operand 2 instruction. The rules of operation for logical addition are illustrated by the following truth table:

Operand 1	Operand 2	Result
0	0	0
0	1	1
1	0	1
1	1	1

The subject instruction is executed as if it were in the normal instruction sequence except that the instruction length code and updated instruction address fields of the current program status word (PSW) reflect the *execute* instruction. The subject instruction itself is never modified permanently in main storage, and the subject instruction cannot be another *execute* instruction.

EX

Normally, instruction sequencing continues with the instruction following the *execute* instruction. However, if the instruction at the operand 2 address is a successful branch instruction, the instruction address field of the current PSW is replaced by the branch address and instruction sequencing continues with the instruction located at the branch address. If the operand 2 instruction is *branch and link* or *branch and link external*, the instruction address stored in the link register is that of the instruction following the *execute* instruction.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	EX	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	EX	$r_1,s_2(x_2)$

Operational Considerations:

- If an interrupt occurs after the completion of the subject instruction, the old PSW contains the address of the instruction following the *execute* instruction or the branch address.
- The condition code may be set by the instruction at the operand 2 address.
- Possible program exception:
  - Specification exception (The address specified by operand 2 is an odd-numbered address.)

NOTE:

A program exception condition can be caused by the execute *instruction or the instruction specified in the* execute *instruction.*

HDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
HDR	24	RR	2

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
■ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the double-word contents of the operand 2 (r_2) register to be divided by 2. The normalized quotient is placed in the double-word operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	HDR	r_1,r_2

Operational Considerations:

- The fraction of operand 2 (r_2) is shifted right one bit position. The least significant bit of the fraction is placed into the most significant bit position of the guard digit, and the vacated fraction bit position is filled with zero. The intermediate result is normalized and placed in the operand 1 (r_1) location.

- When normalization causes the exponent to become less than zero, an exponent underflow condition exists. If the exponent underflow mask bit of the current program status word (PSW) is 1, the exponent of the result is 128 greater than the correct value. If the exponent underflow mask bit of the current PSW is zero, the result is made true zero.

- When the fraction of operand 2 (r_2) is zero, the result is made a true zero, a normalization is not attempted, and a significance exception does not occur.

* *HDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

HER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
HER	**34**	**RR**	**2**

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
■ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the full-word contents of the operand 2 (r_2) register to be divided by 2. The normalized quotient is placed in the full word in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	HER	r_1,r_2

Operational Considerations:

- The fraction of operand 2 (r_2) is shifted right one bit position. The least significant bit of the fraction is placed into the most significant bit position of the guard digit, and the vacated fraction bit position is filled with zero. The intermediate result is normalized and placed in the operand 1 (r_1) location.

- When normalization causes the exponent to become less than zero, an exponent underflow condition exists. If the exponent underflow mask bit of the current program status word (PSW) is 1, the exponent of the result is 128 greater than the correct value. If the exponent underflow mask bit of the current PSW is zero, the result is made true zero.

- When the fraction of operand 2 (r_2) is zero, the result is made a true zero, normalization is not attempted, and a significance exception does not occur.

* *HER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

HPR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
HPR	99	SI	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Alters the current relocation register.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	HPR	$d_1(b_1), i_2$

Implicit Format:

LABEL	△OPERATION△	OPERAND
→ [symbol]	HPR	s_1, i_2

IC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
IC	43	RX	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes one byte from the area in main storage specified by operand 2 to be moved into the least significant eight bits of the operand 1 (r_1) register.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	IC	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	IC	$r_1,s_2(x_2)$

Operational Considerations:

- The contents of operand 2 remain unchanged.
- The contents of the most significant 24 bits of the operand 1 (r_1) register remain unchanged.
- Operand 2 may be an area in main storage defined as longer than one byte, but only one byte will be moved.

ISK

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
ISK	09	RR	2

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Alters the contents and size of the protect key storage.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	ISK	r_1, r_2

L

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
L	58	RX	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of operand 2, a full word in main storage, to be placed in the operand 1 register (r_1).

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	L	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	L	$r_1,s_2(x_2)$

Operational Considerations:

- Operand 2 is a full word in main storage on a full-word boundary.
- The contents of operand 2 remain unchanged.

LA

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LA	41	RX	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Causes the main storage address or the self-defining term specified by operand 2 to be loaded into the least significant 24 bits of the operand 1 (r_1) register. The eight most significant bits of the operand 1 (r_1) register are set to zeros.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LA	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LA	r_1,s_2

Operational Considerations:

- The generated address is not checked for validity.
- The contents of operand 2 remain unchanged.
- If only the x_2 or b_2 register is used and is the same as the operand 1 (r_1) register, the content of the operand 1 (r_1) register is incremented by the decimal value d_2.
- If operand 2 is expressed as a decimal value without the reference of any register, then operand 1 (r_1) is loaded with the operand 2 decimal value.

LCDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LCDR	**23**	**RR**	**2**

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT < 0, SET TO 1
■ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the sign of the double-word contents of the operand 2 (r_2) register to be reversed. The result is placed in the double-word operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LCDR	r_1,r_2

Operational Considerations:

- The exponent and fraction are not changed.
- The contents of operand 2 (r_2) remain unchanged.

* *LCDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LCER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LCER	**33**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the sign of the full-word contents of the operand 2 (r_2) register to be reversed. The result is placed in the full-word operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LCER	r_1,r_2

Operational Considerations:

- The exponent and fraction are not changed.
- The contents of operand 2 (r_2) remain unchanged.

* *LCER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LCR*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LCR	**13**	**RR**	**2**

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT <0, SET TO 1
■ IF RESULT >0, SET TO 2
■ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
■ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the twos complement of the value of the contents of the operand 2 register (r_2) to be placed in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LCR	r_1,r_2

Operational Considerations:

- The twos complement of the second operand is placed in the first operand location.
- A fixed-point overflow condition exists when the maximum negative number is complemented; the number remains unchanged. Zero remains unchanged under complementation.
- Operand 2 (r_2) remains unchanged.

* *LCR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LCS

General

OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LCS	**B1**	**RS**	**4**

Condition Codes

- ■ SET TO 0
- ■ SET TO 1
- □ SET TO 2
- ■ SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Transfers data from main storage to control storage.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LCS	$r_1, r_3, d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LCS	r_1, r_3, s_2

LD*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LD	68	RX	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of a double word in storage specified by operand 2 to be placed in the double word in the operand 1 (r_1) register.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LD	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LD	$r_1,s_2(x_2)$

Operational Consideration:

- The contents of operand 2 remain unchanged.

* *LD is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LDR	28	RR	2

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the double word in the operand 2 (r_2) register to be placed in the double word in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LDR	r_1,r_2

Operational Consideration:

- The contents of operand 2 (r_2) remain unchanged.

* *LDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LE*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LE	78	RX	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a full word in storage specified by operand 2 to be placed in a full word in the operand 1 (r_1) register.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LE	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LE	$r_1,s_2(x_2)$

Operational Consideration:

- The contents of operand 2 remain unchanged.

* *LE is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LER	38	RR	2

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a full word in the operand 2 (r_2) register to be placed in a full word in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LER	r_1,r_2

Operational Consideration:

- The contents of operand 2 (r_2) remain unchanged.

* *LER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LH

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LH	48	RX	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of operand 2, a half word in main storage, to be expanded and placed in the operand 1 register (r_1).

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LH	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LH	$r_1,s_2(x_2)$

Operational Considerations:

- Operand 2 is a half word in main storage on a half-word boundary.
- The contents of operand 2 remain unchanged.
- Operand 2 is placed in the register of operand 1 (r_1) and then is expanded to a full word by propagating the sign bit through the most significant bits.

LM

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LM	98	RS	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes the contents of operand 2, one or more full words in main storage, to be placed in the registers of operand 1 (r_1) through operand 3 (r_3).

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LM	$r_1,r_3,d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LM	r_1,r_3,s_2

Operational Considerations:

- The general registers, starting with the register specified by operand 1 (r_1) and ending with the register specified by operand 3 (r_3), are loaded with full words from main storage, beginning with the address specified by operand 2 (r_2).

- The registers are loaded in ascending numeric sequence, beginning with the register specified by operand 1 (r_1) and continuing through the register specified by operand 3 (r_3).

LM

- One register may be loaded by specifying the same register for both operand 1 (r_1) and operand 3 (r_3).
- If the register specified by operand 3 (r_3) is lower than the register specified by operand 1 (r_1), then the register specified by operand 1 (r_1) and all registers with a number greater than operand 1 (r_1) plus the register specified by operand 3 (r_3) and all registers with a number less than operand 3 (r_3) are loaded.
- The contents of operand 2, in main storage, remain unchanged. Operand 2 must be on a full-word boundary.

LNDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LNDR	**21**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- □ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the sign of the double word in the operand 2 (r_2) register to be made negative. The result is placed in the double-word register specified by operand 1 (r_1).

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LNDR	r_1, r_2

Operational Considerations:

- Operand 2 (r_2) is made negative even if the fraction is zero.
- The exponent and fraction are not changed.
- The contents of operand 2 (r_2) remain unchanged.

* *LNDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LNER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LNER	**31**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- □ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the sign of a full word in the operand 2(r_2) register to be made negative. The result is placed in a full word in the register specified by operand 1 (r_1).

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LNER	r_1,r_2

Operational Considerations:

- Operand 2 (r_2) is made negative even if the fraction is zero.
- The exponent and fraction are not changed.
- The contents of operand 2 (r_2) remain unchanged.

* *LNER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LNR*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LNR	**11**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- □ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- □ SIGNIFICANCE
- □ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the twos complement of the absolute value of the contents of the operand 2 and register (r_2) to be placed in the operand 1 (r_1) register.

Explicit and Implicit:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LNR	r_1,r_2

Operational Considerations:

- The twos complement of the absolute value of the second operand (r_2) is placed in the first operand (r_1) location.
- The operation complements positive numbers; negative numbers and zero remain unchanged.
- Operand 2 (r_2) remains unchanged.

* *LNR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LPDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LPDR	**20**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- □ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the sign of the double word in the operand 2 (r_2) register to be positive. The result is placed in the double word of the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LPDR	r_1,r_2

Operational Considerations:

- The exponent and fraction are not changed.
- The contents of operand 2 (r_2) remain unchanged.

* *LPDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LPER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LPER	**30**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- □ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions	
□ ADDRESSING	□ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the sign of a full word in the operand 2 (r_2) register to be positive. The result is placed in a full word of the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LPER	r_1,r_2

Operational Considerations:

- The exponent and fraction are not changed.
- The contents of operand 2 (r_2) remain unchanged.

* *LPER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LPR*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LPR	10	RR	2

Condition Codes
■ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
■ IF RESULT > 0, SET TO 2
■ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
■ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the absolute value of the contents of the operand 2 register (r_2) to be placed in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LPR	r_1,r_2

Operational Considerations:

- Positive numbers remain unchanged. When the second operand (r_2) is negative, the twos complement is placed in the first operand (r_1) location.
- A fixed-point overflow condition exists and the number remains unchanged when the maximum negative number is complemented.
- Operand 2 (r_2) remains unchanged.

* *LPR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LPSW

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LPSW	82	SI	4

Condition Codes

- ■ SET TO 0
- ■ SET TO 1
- ■ SET TO 2
- ■ SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- □ SIGNIFICANCE
- ■ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - ■ OP 1 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Replaces all or part of the current PSW.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LPSW	$d_1(b_1), i_2$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LPSW	s_1, i_2

LR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LR	18	RR	2

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	■ NONE

Function:

Causes the contents of the register specified by operand 2 (r_2) to be loaded into the register specified by operand 1 (r_1).

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LR	r_1,r_2

Operational Considerations:

- The contents of the register specified by operand 2 (r_2) are loaded into the register specified by operand 1 (r_1).

- The contents of the register specified by operand 2 (r_2) remain unchanged.

LTDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LTDR	**22**	**RR**	**2**

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT < 0, SET TO 1
■ IF RESULT > 0, SET TO 2
□ IF OVERFLOW, SET TO 3
□ UNCHANGED

Possible Program Exceptions	
□ ADDRESSING	□ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the double-word contents of the operand 2 (r_2) register to be placed in the double-word operand 1 (r_1) register. The condition code is set by this instruction.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LTDR	r_1,r_2

Operational Considerations:

- The contents of operand 2 (r_2) remain unchanged.
- When the same register is specified by operand 1 (r_1) and operand 2 (r_2), the operation is equivalent to a test without data movement.

* *LTDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LTER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LTER	**32**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a full word in the operand 2 (r_2) register to be placed in a full word in the operand 1 (r_1) register. The condition code is set by this instruction.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	LTER	r_1,r_2

Operational Considerations:

- The contents of operand 2 (r_2) remain unchanged.
- When the same register is specified by operand 1 (r_1) and operand 2 (r_2), the operation is equivalent to a test without data movement.

* *LTER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

LTR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
LTR	12	RR	2

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT <0, SET TO 1
■ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	■ NONE

Function:

Causes the contents of the register specified by operand 2 (r_2) to be loaded into the register specified by operand 1 (r_1) and the condition code to be set to reflect the value contained in the registers.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	LTR	r_1,r_2

Operational Considerations:

- The contents of the register specified by operand 2 (r_2) are loaded into the register specified by operand 1 (r_1).
- The contents of the register specified by operand 2 (r_2) remain unchanged.

M

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
M	5C	RX	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	■ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes the contents of the odd register of the even-odd pair specified by operand 1(r_1) to be multiplied by the contents of operand 2, a full word in main storage. The product is placed in the even-odd pair of registers specified by operand 1 (r_1).

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	M	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	M	$r_1,s_2(x_2)$

Operational Considerations:

- Both operands are treated as fixed-point, 32-bit signed integers.
- The contents of operand 2, the multiplier in a full word in main storage, remain unchanged.
- The product is treated as a 64-bit, fixed-point signed integer and occupies and even-odd register pair specified by operand 1 (r_1).

M

- The multiplicand is first loaded into the odd-numbered register of the even-odd pair specified by operand 1 (r_1). The content of the even-numbered register is ignored until replaced by the most significant 32 bits of the product.

- The sign of the product is determined algebraically.

- A specification exception results if operand 2 is not on a full-word boundary and also if operand 1 (r_1) specifies an odd-numbered register.

MD*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MD	**6C**	**RX**	**4**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ■ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the double word in the operand 1 (r_1)register to be multiplied by the contents of a double word in main storage specified by operand 2. The normalized product is placed in the double word of the operand 1 (r_1) register.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MD	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MD	$r_1,s_2(x_2)$

* *MD is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

MDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MDR	**2C**	**RR**	**2**

Condition Codes
□ IF RESULT = 0, SET TO 0
□ IF RESULT < 0, SET TO 1
□ IF RESULT > 0, SET TO 2
□ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
□ ADDRESSING	□ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
■ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the contents of the double word in the operand 1 (r_1) register to be multiplied by the contents of the double word in the operand 2 (r_2) register. The normalized product is placed in the double word of the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MDR	r_1,r_2

* *MDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

ME*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
ME	**7C**	**RX**	**4**

Condition Codes
□ IF RESULT = 0, SET TO 0
□ IF RESULT < 0, SET TO 1
□ IF RESULT > 0, SET TO 2
□ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
■ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the contents of a full word in the operand 1 (r_1) register to be multiplied by the contents of a full word in main storage specified by operand 2. The normalized product is placed in a full word of the operand 1 (r_1) register.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	ME	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	ME	$r_1,s_2(x_2)$

* *ME is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

MER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MER	**3C**	**RR**	**2**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ■ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a full word in the operand 1 (r_1) register to be multiplied by the contents of a full word in the operand 2 (r_2) register. The normalized product is placed in a full word in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MER	r_1,r_2

* *MER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MH	4C	RX	4

Condition Codes
□ IF RESULT = 0, SET TO 0
□ IF RESULT < 0, SET TO 1
□ IF RESULT > 0, SET TO 2
□ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	□ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	■ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the contents of the register specified by operand 1 (r_1) to be multiplied by the contents of operand 2, a half word in main storage. The product is placed in the register specified by operand 1 (r_1).

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MH	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MH	$r_1,s_2(x_2)$

Operational Considerations:

- Operand 2 is expanded after being read from storage; then both operands are treated as fixed-point, 32-bit signed integers.
- The contents of operand 2, the multiplier, a half word in main storage, remain unchanged.
- The sign of the product is determined algebraically.
- If the multiplication results in a product that exceeds 32 bits, the high-order bits are ignored but the overflow condition is not indicated. The sign and value of the product may not be correct after overflow.
- A specification exception will result if operand 2 is not on a half-word boundary.

* *MH is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

MP

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MP	FC	SS	6

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
■ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes the contents of operand 1 to be multiplied by the contents of operand 2. The product is placed in the operand 1 location.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	MP	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	MP	$s_1(l_1),s_2(l_2)$

Operational Considerations:

- All signs and digits are checked for validity, and the sign of the product is determined algebraically.
- Operand 1 must be longer than operand 2.
- Operand 1 and operand 2 may overlap if their least significant bytes coincide.
- The size of the multiplier (operand 2) cannot be more than 15 digits and sign.

MP

- The number of digits in the product is equal to the number of digits in the operands; therefore, the multiplicand (operand 1) must have a field of most significant zero digits to equal, in size, operand 2. The maximum product size is 31 digits. At least one most significant digit of the product field is zero.

- Data exception indicates one or more of the following conditions:

  — Invalid sign or digit code

  — Operand 1 has insufficient high-order zero digits

  — Incorrect overlap

MR*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MR	1C	RR	2

Condition Codes

☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions

☐ ADDRESSING
☐ DATA (INVALID SIGN/DIGIT)
☐ DECIMAL DIVIDE
☐ DECIMAL OVERFLOW
☐ EXECUTE
☐ EXPONENT OVERFLOW
☐ EXPONENT UNDERFLOW
☐ FIXED-POINT DIVIDE
☐ FIXED-POINT OVERFLOW
☐ FLOATING-POINT DIVIDE
■ OPERATION
☐ PROTECTION
☐ SIGNIFICANCE
■ SPECIFICATION:
☐ NOT A FLOATING-POINT REGISTER
☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
■ OP 1 NOT EVEN NUMBERED REGISTER
☐ OP 1 NOT ODD NUMBERED REGISTER
☐ NONE

Function:

Causes the contents of the odd register of the even-odd pair specified by operand 1 (r_1) to be multiplied by the contents of the register specified by operand 2 (r_2). The product is placed in the even-odd pair of registers specified by operand 1 (r_1).

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MR	r_1,r_2

Operational Considerations:

- Both operands are treated as fixed-point, 32-bit signed integers.
- The contents of operand 2 (r_2), the multiplier, remain unchanged.
- The product is treated as a 64-bit, fixed-point signed integer and occupies an even-odd register pair specified by operand 1 (r_1).
- The multiplicand is first loaded into the odd-numbered register of the even-odd pair specified by operand 1 (r_1). The content of the even-numbered register is ignored until replaced by the most significant 32 bits of the product.
- The sign of the product is determined algebraically.
- A specification exception results if operand 1 (r_1) specifies an odd-numbered register.

* *MR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

MVC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MVC	D2	SS	6

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes the contents of the field in main storage specified by operand 2 to be placed in the field in main storage specified by operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MVC	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MVC	$s_1(l),s_2$

Operational Considerations:

- The transfer proceeds from left to right.
- The number of bytes transferred is specified by 1 in operand 1.
- The contents of operand 2 remain unchanged unless operand 1 and operand 2 overlap.
- If the number of bytes to be moved is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

MVI

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MVI	92	SI	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the one byte of data used in the instruction as operand 2 to be moved into the one byte of main storage specified by operand 1.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	MVI	$d_1(b_1),i_2$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	MVI	s_1,i_2

Operational Considerations:

- The immediate data in the instruction, operand 2, must specify one byte of data.
- The length attribute of the field specified by operand 1 may be longer than one byte, but only the one byte addressed by operand 1 will be replaced by the immediate data (operand 2).

MVN

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MVN	**D1**	**SS**	**6**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the least significant four bits (the digit or numeric field) of each byte specified by operand 2 to be moved to the least significant four bits of each byte of operand 1.

Explicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	MVN	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	MVN	$s_1(l),s_2$

Operational Considerations:

- The four most significant bits of each byte (zone field) remain unchanged.
- The contents of operand 2 remain unchanged unless there is overlapping.
- Overlapping of operands is permitted.
- The number of bytes transferred is specified by l in operand 1.
- If the number of bytes to be moved is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

MVO

General				
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MVO	**F1**	**SS**	**6**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Moves the contents of operand 2 to operand 1 with a 4-bit (half-byte) shift to the left.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MVO	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MVO	$s_1(l_1),s_2(l_2)$

Operational Considerations:

- This instruction proceeds from right to left.
- The operands are not checked for valid codes.
- Overlapping fields may occur. Unless the operands overlap, operand 2 and the least significant four bits of operand 1 remain unchanged.
- If the second operand is exhausted before the first operand, the remaining first operand field is zero filled. If the result exceeds the capacity of the first operand field, the remaining digits of the second operand are ignored. This operation, in effect, prefixes the least significant digit or sign of the first operand with the digits of the second operand.

MVZ

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
MVZ	**D3**	**SS**	**6**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the most significant four bits (the zone field) of each byte specified by operand 2 to be moved to the most significant four bits of each byte of operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MVZ	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	MVZ	$s_1(l),s_2$

Operational Considerations:

- The four least significant bits of each byte (digit field) remain unchanged.
- The contents of operand 2 remain unchanged unless there is overlapping.
- Overlapping of operands is permitted.
- The number of bytes transferred is specified by l in operand 1.
- If the number of bytes to be moved is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

N

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
N	54	RX	4

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT ≠ 0, SET TO 1
□ IF RESULT > 0, SET TO 2
□ IF OVERFLOW, SET TO 3
□ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	□ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
□ OPERATION	□ NONE

Function:

Causes a logical full-word AND operation to be performed on the contents of operand 1 (r_1) and operand 2. The result is stored in the operand 1 (r_1) register. Operand 2 is a full word in main storage.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	N	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	N	$r_1,s_2(x_2)$

Operational Considerations:

- If the corresponding bit positions in both operand 1 and operand 2 contain 1, the resultant bit will be 1. If either bit is zero, the resultant bit will be zero.

- The rules of operation for logical AND (N) are illustrated by the following truth table:

N

Operand 1	Operand 2	Result (Operand 1)
0	0	0
0	1	0
1	0	0
1	1	1

- It is possible to clear selected bits in operand 1 (r_1) by specifying zeros in the corresponding bit positions of operand 2.

- Operand 2 must be on a full-word boundary.

NC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
NC	D4	SS	6

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT ≠0, SET TO 1
- □ IF RESULT >0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- □ OPERATION
- ■ PROTECTION
- □ SIGNIFICANCE
- □ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes a logical AND operation to be performed on the contents of operand 1 and operand 2. Both operands are located in main storage. The result is stored in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	NC	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	NC	$s_1(l),s_2$

Operational Considerations:

- If the corresponding bit positions in both operand 1 and operand 2 contain 1, the resultant bit will be 1. If either bit is zero, the resultant bit will be zero.
- The rules of operation for logical AND (NC) are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
0	1	0
1	0	0
1	1	1

- It is possible to clear selected bits in operand 1 by specifying zeros in the corresponding bit positions of operand 2.
- The number of bytes involved in the AND instruction is specified by l in operand 1.
- If the number of bytes to be used is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

NI

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
NI	94	SI	4

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT ≠ 0, SET TO 1
□ IF RESULT > 0, SET TO 2
□ IF OVERFLOW, SET TO 3
□ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	□ SPECIFICATION:
□ DECIMAL OVERFLOW	□ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
□ OPERATION	□ NONE

Function:

Causes a logical AND operation to be performed on the contents of operand 1, a byte in main storage, and operand 2, a byte of immediate data in the instruction. The result is stored in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	NI	$d_1(b_1),i_2$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	NI	s_1,i_2

Operational Considerations:

- If the corresponding bit positions in both operand 1 and operand 2 contain 1, the resultant bit will be 1. If either bit is zero, the resultant bit will be zero.

- The rules of operation for logical AND (NI) are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
0	1	0
1	0	0
1	1	1

- It is possible to clear selected bits in operand 1 by specifying zeros in the corresponding bit positions of operand 2.

NR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
NR	14	RR	2

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT ≠ 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	■ NONE

Function:

Causes a logical AND operation to be performed on the contents of the registers specified by operand 1 (r_1) and operand 2 (r_2). The result is stored in operand 1 (r_1).

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	NR	r_1,r_2

Operational Considerations:

- If the corresponding bit positions in both operand 1 (r_1) and operand 2 (r_2) contain 1, the resultant bit will be 1. If either bit is zero, the resultant bit will be zero.

- The rules of operation for logical AND (NR) are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
0	1	0
1	0	0
1	1	1

- It is possible to clear selected bits in operand 1 by specifying zeros in the corresponding bit positions of operand 2.

O

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
O	56	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT ≠0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes a logical OR operation to be performed on the contents of operand 1 (r_1) and operand 2, a full word in main storage. The result is stored in operand 1 (r_1).

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	O	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	O	$r_1,s_2(x_2)$

Operational Considerations:

- A bit position in the result is set to 1 if the corresponding bit positions in either or both operands contain 1; otherwise, the result bit position is set to zero.

- The rules of operation for logical OR (O) are illustrated by the following truth table:

O

Operand 1	Operand 2	Result (Operand 1)
0	0	0
0	1	1
1	0	1
1	1	1

- Operand 2 must be on a full-word boundary.

OC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
OC	D6	SS	6

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT ≠ 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes a logical OR operation to be performed on the contents of main storage specified by operand 1 and operand 2. The result is stored in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	OC	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	OC	$s_1(l),s_2$

Operational Considerations:

- A bit position in the result is set to 1 if the corresponding bit positions in either or both operands contain 1; otherwise, the result bit position is set to zero.

- The rules of operation for logical OR (OC) are illustrated by the following truth table:

OC

Operand 1	Operand 2	Results (Operand 1)
0	0	0
0	1	1
1	0	1
1	1	1

- The number of bytes used is specified by l in operand 1.
- If the number of bytes to be used is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

OI

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
OI	96	SI	4

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT ≠ 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes a logical OR operation to be performed on the contents of operand 1 (a byte in main storage) and operand 2 (a byte of immediate data in the instruction). The result is stored in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	OI	$d_1(b_1),i_2$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	OI	s_1,i_2

Operational Considerations:

- A bit position in the result is set to 1 if the corresponding bit positions in either or both operands contain 1; otherwise, the result bit position is set to zero.

OI

- The rules of operation for logical OR (OI) are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
0	1	1
1	0	1
1	1	1

OR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
OR	16	RR	2

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT ≠ 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Causes a logical OR operation to be performed on the contents of the registers specified by operand 1 (r_1) and operand 2 (r_2). The result is stored in operand 1 (r_1).

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	OR	r_1,r_2

Operational Considerations:

- A bit position in the result is set to 1 if the corresponding bit positions in either or both operands contain 1; otherwise, the result bit position is set to zero.
- The rules of operation for logical OR (OR) are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
0	1	1
1	0	1
1	1	1

PACK

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
PACK	F2	SS	6

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Converts the contents of operand 2 from the unpacked format to the packed format, which is placed in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	PACK	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	PACK	$s_1(l_1),s_2(l_2)$

Operational Considerations:

- This instruction proceeds one byte at a time from right to left. The first byte operated on has its sign and digit reversed. (An F4 becomes 4F.) Each byte from then on has its zone removed and the digit half of the byte packed into the receiving area.
- If operand 2 does not completely fill operand 1, the remaining operand 1 field is zero filled.
- If the result exceeds the capacity of the operand 1 field, the remaining operand 2 digits are ignored.
- The operands are not checked for valid codes.
- Overlapping fields may occur; each resultant byte is processed after each operand byte.

S

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
S	5B	RX	4

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT < 0, SET TO 1
■ IF RESULT > 0, SET TO 2
■ IF OVERFLOW, SET TO 3
□ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	□ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
■ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
□ OPERATION	□ NONE

Function:

Causes the contents of operand 2, a full word in main storage, to be subtracted from the contents of the register specified by operand 1 (r_1). The results are placed in the operand 1 (r_1) register.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	S	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	S	$r_1,s_2(x_2)$

Operational Considerations:

- The subtraction is performed by converting the number in operand 2 into a signed twos complement binary number and then algebraically adding it to the value in operand 1 (r_1).
- The maximum fixed-point number that can be contained in a 32-bit register is 2,147,483,647($2^{31}-1$). The minimum number is –2,147,483,648(-2^{31}). For decimal numbers outside this range, an overflow condition is produced.
- Operand 2 must be on a full-word boundary.
- The contents of operand 2 are not changed by the subtract (S) instruction.

SD*

Floating Point

General				Possible Program Exceptions	
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)	■ ADDRESSING	■ PROTECTION
MNEM.	HEX.			□ DATA (INVALID SIGN/DIGIT)	■ SIGNIFICANCE
SD	6B	RX	4	□ DECIMAL DIVIDE	■ SPECIFICATION:
				□ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
				□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
				■ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
				■ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
				□ FIXED-POINT DIVIDE	■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
				□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
				□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
				■ OPERATION	□ NONE

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT < 0, SET TO 1
■ IF RESULT > 0, SET TO 2
□ IF OVERFLOW, SET TO 3
□ UNCHANGED

Function:

Causes the contents of a double word in main storage specified by operand 2 to be algebraically subtracted from the contents of the double word register specified by operand 1 (r_1). The normalized difference is placed in the operand 1 (r_1) register.

Explicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	SD	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	SD	$r_1,s_2(x_2)$

Operational Consideration:

- The execution of the SD instruction is identical to that of the AD instruction except that the sign of operand 2 is reversed before addition.

* *SD is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SDR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SDR	**2B**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- ■ EXPONENT OVERFLOW
- ■ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the contents of the double-word register specified by operand 2 (r_2) to be algebraically subtracted from the contents of the double-word register specified by operand 1 (r_1). The normalized difference is placed in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SDR	r_1,r_2

Operational Consideration:

- The execution of the SDR instruction is identical to that of the ADR instruction, except that the sign of operand 2 (r_2) is reversed before addition.

* *SDR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SE*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SE	7B	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ■ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a full word in main storage specified by operand 2 to be algebraically subtracted from a full word in the register specified by operand 1 (r_1). The normalized difference is placed in the operand 1 (r_1) register.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SE	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SE	$r_1,s_2(x_2)$

Operational Consideration:

- The execution of the SE instruction is identical to that of the AE instruction, except that the sign of operand 2 is reversed before addition.

* *SE is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SER*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SER	**3B**	**RR**	**2**

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT <0, SET TO 1
■ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	■ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
■ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of a full word in the operand 2 (r_2) register to be algebraically subtracted from a full word in the operand 1 (r_1) register. The normalized difference is placed in a full word in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SER	r_1,r_2

Operational Consideration:

- The execution of the SER instruction is identical to that of the AER instruction, except that the sign of operand 2 is reversed before addition.

* *SER is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SH

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SH	4B	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ■ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of operand 2, a half word in main storage, to be subtracted from the contents of the register specified by operand 1 (r_1). The results are to be placed in the operand 1 (r_1) register.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SH	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SH	$r_1,s_2(x_2)$

Operational Considerations:

- The subtraction is performed by converting the number in operand 2 into a signed twos complement binary number, expanded to a full word, and then algebraically adding it to the value in operand 1 (r_1).
- The maximum fixed-point number that can be contained in 32-bit register is 2,147,483,647(2^{31}—1); the minimum number is —2,147,483,648(—2^{31}). For decimal numbers outside this range, an overflow condition is produced.
- Operand 2 must be on a half-word boundary.
- The contents of operand 2 are not changed by the *subtract half word* (SH) instruction.

SIO

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SIO	9C	SI	4

Condition Codes

- ■ SET TO 0
- ■ SET TO 1
- ■ SET TO 2
- ■ SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Initiates input and output operations to be executed by the I/O channels and the I/O status tabler.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SIO	$d_1(b_1)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SIO	s_1

SL*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SL	5F	RX	4

Condition Codes

- ☐ SET TO 0
- ■ SET TO 1
- ■ SET TO 2
- ■ SET TO 3
- SEE OPER. CONSIDERATIONS

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a full word in main storage specified by operand 2 to be subtracted logically from the contents of the operand 1 (r_1) register. The difference is placed in operand 1 (r_1).

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SL	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SL	$r_1,s_2(x_2)$

Operational Considerations:

- The subtraction is performed by adding the twos complement of operand 2 to operand 1.
- All 32 bits of both operands are used.
- The contents of operand 2 remain unchanged.
- Operand 2 must be on a full-word boundary.

* *SL is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SL

- The condition code is set:
  - to 1 if result is not zero (no carryout of most significant bit position);
  - to 2 if result is zero (carryout of most significant bit position); or
  - to 3 if result is not zero (carryout of most significant bit position).

  Code 0 is not used. A zero difference cannot be obtained without a carryout of the most significant bit position.

SLA*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SLA	8B	RS	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- ■ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- □ SIGNIFICANCE
- □ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the 31-bit integer field in the register specified by operand 1 (r_1) to be shifted left the number of bit positions specified by the six low-order bits of the second operand (s_2) address.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SLA	$r_1,d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SLA	r_1,s_2

Operational Considerations:

- The 31-bit integer of the first operand (r_1) is shifted left the number of bit positions specified by the low-order six bits of the second operand address.
- The vacated low-order bit positions of the register are zero filled. The sign bit of the register remains unchanged.
- If a bit unlike the sign bit is shifted out of the high-order numeric bit position, a fixed-point overflow condition exists.

* *SLA is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SLA

- 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.
- A shift of 31 bits causes the entire integer to be shifted out of the register. When the entire integer field for a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -2^{31}.
- A zero shift value provides a sign and magnitude test.

SLDA*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SLDA	8F	RS	4

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT <0, SET TO 1
■ IF RESULT >0, SET TO 2
■ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
■ FIXED-POINT OVERFLOW	■ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the 63-bit integer field in the pair of registers specified by operand 1 (r_1) to be shifted left the number of bit positions specified by the six low-order bits of the second operand (s_2) address.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SLDA	$r_1,d_2(b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SLDA	r_1,s_2

Operational Considerations:

- Operand 1 (r_1) must refer to an even-numbered register of an even-odd register pair.
- The contents of both registers, except the sign bit of the even register, are shifted as one 63-bit integer. The vacated low-order bit positions of the odd register are zero filled. The sign bit of the even register remains unchanged.
- If a bit unlike the sign bit is shifted out of the high-order numeric bit position of the even register, a fixed-point overflow condition exists.

* *SLDA is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SLDA

- A zero shift value in the double-shift operations provides a double-length sign and magnitude test.
- 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.
- Shifting 63 bits causes the entire integer to be shifted out of the registers. When the entire integer field for a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -2^{31}.

SLDL*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SLDL	8D	RS	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	■ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of the double word in the pair of registers specified by operand 1 (r_1) to be shifted left the number of bit positions specified by the least significant six bits of the operand 2 address.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SLDL	$r_1,d_2(b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SLDL	r_1,s_2

Operational Considerations:

- The vacated least significant bit positions of the registers are zero filled.
- Bits shifted out of the even-numbered register are lost.
- Operand 1 (r_1) must refer to the even-numbered register of an even-odd register pair.

* *SLDL is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SLL

General				Possible Program Exceptions	
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)	☐ ADDRESSING	☐ PROTECTION
MNEM.	HEX.			☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
SLL	89	RS	4	☐ DECIMAL DIVIDE	☐ SPECIFICATION:
				☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
				☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
				☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
				☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
				☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
				☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
				☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
				☐ OPERATION	■ NONE

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Function:

Causes a full word in operand 1 (r_1) to be shifted left the number of bit positions specified by the least significant six bits of the operand 2 address.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SLL	$r_1,d_2(b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SLL	r_1,s_2

Operational Considerations:

- The vacated least significant bit positions of the register are zero filled.
- Bits shifted out of the register are lost.

SLM

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SLM	**B8**	**RS**	**4**

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of operand 2, one or more full words in main storage, to be placed in the problem registers of operand 1 (r_1) through operand 3 (r_3).

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SLM	$r_1,r_3,d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SLM	r_1,r_3,s_2

SLR*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SLR	1F	RR	2

Condition Codes

- ☐ SET TO 0
- ■ SET TO 1
- ■ SET TO 2
- ■ SET TO 3

SEE OPER. CONSIDERATIONS

Possible Program Exceptions

☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of the operand 2 (r_2) register to be subtracted logically from the contents of the operand 1 (r_1) register. The difference is placed in operand 1 (r_1).

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SLR	r_1,r_2

Operational Considerations:

- The subtraction is performed by adding the twos complement of operand 2 to operand 1.
- All 32 bits of both operands are used.
- The contents of operand 2 remain unchanged.
- The condition code is set:
  - to 1 if result is not zero (no carryout of most significant bit position);
  - to 2 if result is zero (carryout of most significant bit position); or
  - to 3 if result is not zero (carryout of most significant bit position).

  Code 0 is not used. A zero difference cannot be obtained without a carryout of the most significant bit position.

* *SLR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SP

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SP	FB	SS	6

Condition Codes
■ IF RESULT = 0, SET TO 0
■ IF RESULT <0, SET TO 1
■ IF RESULT >0, SET TO 2
■ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
■ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
■ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Subtracts the contents of operand 2 from the contents of operand 1. The results are placed in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SP	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SP	$s_1(l_1),s_2(l_2)$

Operational Considerations:

- Subtraction is accomplished by reversing the sign of operand 2 and performing a decimal add. The contents and sign of operand 2 are not affected by this operation.
- All signs and digits are checked for validity and the sign of the result is determined algebraically.
- A zero result has a positive sign when the operation is completed without overflow.
- When most significant digits are lost because of overflow, the partial result has the sign that the correct result would have had.

SP

- If operand 2 is shorter than operand 1, operand 2 is extended with zero digits.
- An overflow condition results if the capacity of the operand 1 field is exceeded by the result or if the carryout of the most significant digit position of the result field is lost.
- Operand 1 and operand 2 may overlap if their least significant bytes coincide. Incorrect overlay will cause a data exception.

SPM

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SPM	04	RR	2

Condition Codes

- ■ SET TO 0
- ■ SET TO 1
- ■ SET TO 2
- ■ SET TO 3
- ☐ SEE OPER. CONSIDERATIONS

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Causes the program mask field (bits 34 through 39) of the current program status word (PSW) to be changed according to the contents of operand 1 (r_1).

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SPM	r_1

Operational Considerations:

- Bits 2 through 7 of the full-word contents of operand 1 (r_1) replace the program mask field (bits 34 through 39) of the current PSW.
- Bits 0, 1, and 8 through 31 of r_1 are ignored.
- The condition code is set equal to bit positions 2 and 3 of operand 1.

SR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SR	1B	RR	2

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ■ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the operand 2 (r_2) register to be subtracted from the contents of the operand 1 (r_1) register. The results are placed in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SR	r_1,r_2

Operational Considerations:

- The subtraction is performed by converting the number in operand 2 (r_2) into a signed twos complement binary number and then algebraically adding it to the value in operand 1 (r_1).
- The maximum fixed-point number that can be contained in a 32-bit register is 2,147,483,647($2^{31}-1$); the minimum number is —2,147,483,648(-2^{31}). For decimal numbers outside this range, an overflow condition is produced.
- The contents of operand 2 (r_2) are not changed by the subtract (SR) instruction.

SRA*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SRA	8A	RS	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the 31-bit integer field in the register specified by operand 1 (r_1) to be shifted right the number of bit positions specified by the six lower bits of the second operand (s_2) address.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SRA	$r_1,d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SRA	r_1,s_2

Operational Considerations:

- The 31-bit integer field of the first operand (r_1) is shifted right the number of bit positions specified by the low-order six bits of the second operand address. The sign bit remains unchanged.
- The bits shifted out of the low-order bit position of the register are lost; the vacated high-order bit positions of the register are sign filled.

* *SRA is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

- 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. For example, 5 shifted right by one bit position yields +2, whereas —5 yields —3.

- A shift of 31 bits causes 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 zero. For a negative number, the register contains a value of —1.

- A zero shift value provides a sign and magnitude test.

SRDA*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SRDA	**8E**	**RS**	**4**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT $<$0, SET TO 1
- ■ IF RESULT $>$0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ■ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the 63-bit integer field in the pair of registers specified by operand 1 (r_1) to be shifted right the number of bit positions specified by the six low-order bits of the second operand (s_2) address.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SRDA	$r_1,d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SRDA	r_1,s_2

Operational Considerations:

- Operand 1 (r_1) must refer to an even-numbered register of an even-odd register pair.
- The contents of both registers, except the sign bit of the even register, are shifted as one 63-bit integer. The bits shifted out of the low-order bit position of the odd register are lost; the vacated high-order bit positions of the register pair are sign filled.
- A right shift of one bit position is equivalent to dividing the number by 2, without a remainder.
- Shifting 63 bits causes the entire integer to be shifted out of the register. When the entire integer field for a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of —1.
- A zero shift value in the double-shift operations provides a double-length sign and magnitude test.

* *SRDA is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SRDL*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SRDL	8C	RS	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ■ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the double word in the pair of registers specified by operand 1 (r_1) to be shifted right the number of bit positions specified by the least significant six bits of the operand 2 address.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SRDL	$r_1,d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SRDL	r_1,s_2

Operational Considerations:

- The vacated most significant bit positions of the registers are zero filled.
- Bits shifted out of the odd-numbered register are lost.
- Operand 1 (r_1) must refer to the even-numbered register of an even-odd register pair.

* *SRDL is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SRL

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SRL	88	RS	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Causes a full word in operand 1 (r_1) to be shifted right the number of bit positions specified by the least significant six bits of the operand 2 address.

Explicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	SRL	$r_1,d_2(b_2)$

Implicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	SRL	r_1,s_2

Operational Considerations:

- The vacated most significant bit positions of the register are zero filled.
- Bits shifted out of the register are lost.

SSFS

General

OPCODE MNEM.	OPCODE HEX.	FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
SSFS	**A2**	**RS**	**4**

Condition Codes

■ IF RESULT = 0, SET TO 0
☐ IF RESULT <0, SET TO 1
☐ IF RESULT >0, SET TO 2
■ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions

■ ADDRESSING
☐ DATA (INVALID SIGN/DIGIT)
☐ DECIMAL DIVIDE
☐ DECIMAL OVERFLOW
☐ EXECUTE
☐ EXPONENT OVERFLOW
☐ EXPONENT UNDERFLOW
☐ FIXED-POINT DIVIDE
☐ FIXED-POINT OVERFLOW
☐ FLOATING-POINT DIVIDE
■ OPERATION

■ PROTECTION
☐ SIGNIFICANCE
■ SPECIFICATION:
☐ NOT A FLOATING-POINT REGISTER
☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
■ OP 1 NOT EVEN NUMBERED REGISTER
☐ OP 1 NOT ODD NUMBERED REGISTER
☐ NONE

Function:

Samples data at a specified rate after a sync pattern has been detected on the selected SOFTSCOPE data bus.

Explicit and Implicit Format:

The bit pattern is the format of the instruction.

SSK

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SSK	08	RR	2

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Specifies storage protection blocks of 512 bytes or 1024 bytes.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SSK	r_1, r_2

SSM

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SSM	**80**	**SI**	**4**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ■ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the system mask of the current PSW to be replaced by the first half word of the first operand (bits 0—7).

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SSM	$d_1(b_1)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SSM	s_1

SSRS

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SSRS	**A3**	**RS**	**4**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	■ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Samples data at a specified rate, and stores the data into a revolving buffer until a sync pattern has been detected on the selected SOFTSCOPE data bus, or until the internal timer lapses.

Explicit and Implicit Format:

The bit pattern is the format of the instruction.

SSTM

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SSTM	**B0**	**RS**	**4**

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the registers specified by operand 1 (r_1) through operand 3 (r_3) to be stored in operand 2, one or more full words in main storage.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SSTM	$r_1,r_3,d_2(b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SSTM	r_1,r_3,s_2

ST

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
ST	50	RX	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes the contents of the operand 1 (r_1) register to be stored in operand 2, a full word in main storage.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	ST	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	ST	$r_1,s_2(x_2)$

Operational Considerations:

- The contents of the operand 1 (r_1) register are not changed by the *store* (ST) instruction.
- Operand 2, a full word in main storage, must be on a full-word boundary.
- Operand 1 is the sending field, operand 2 the receiving field.

STC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
STC	42	RX	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the least significant eight bits of the operand 1 (r_1) register to be stored in a byte of main storage specified by operand 2.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	STC	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	STC	$r_1,s_2(x_2)$

Operational Considerations:

- The contents of operand 1 (r_1) remain unchanged.

STD*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
STD	**60**	**RX**	**4**

Condition Codes
□ IF RESULT = 0, SET TO 0
□ IF RESULT <0, SET TO 1
□ IF RESULT >0, SET TO 2
□ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	■ SPECIFICATION:
□ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the contents of the register specified by operand 1 (r_1) to be placed in a double word in main storage specified by operand 2.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	STD	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	STD	$r_1,s_2(x_2)$

Operational Considerations:

- The contents of the operand 1 (r_1) register remain unchanged.

* *STD is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

STE*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
STE	**70**	**RX**	**4**

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of a full word in the register specified by operand 1 (r_1) to be placed in a full word in main storage specified by operand 2.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	STE	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	STE	$r_1,s_2(x_2)$

Operational Consideration:

- The contents of the operand 1 (r_1) register remain unchanged.

* *STE is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

STH

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
STH	40	RX	4

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
■ UNCHANGED

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	■ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Causes the least significant 16 bits of the operand 1 (r_1) register to be stored in operand 2, a half word in main storage.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	STH	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	STH	$r_1,s_2(x_2)$

Operational Considerations:

- The contents of the operand 1 (r_1) register are not changed by the *store half word* (STH) instruction.
- Operand 2, a half word in main storage, must be on a half-word boundary.
- Operand 1 is the sending field, operand 2 the receiving field.

STM

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
STM	90	RS	4

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the registers specified by operand 1 (r_1) through operand 3 (r_3) to be stored in operand 2, one or more full words in main storage.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	STM	$r_1,r_3,d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	STM	r_1,r_3,s_2

Operational Considerations:

- The contents of the general registers starting with the register specified by operand 1 (r_1) and ending with the register specified by operand 3 (r_3) are stored in one or more full words in main storage beginning with the address specified by operand 2 (s_2).
- The registers are used in ascending numeric sequence beginning with the register specified by operand 1 (r_1) and continuing through the register specified by operand 3 (r_3).
- One register may be stored by specifying the same register for both operand 1 (r_1) and operand 3 (r_3).

STM

- If the register specified by operand 3 (r_3) is lower than the register specified by operand 1 (r_1) then the register specified by operand 1 (r_1) and all registers with a number greater than operand 1 (r_1), plus the register specified by operand 3 (r_3) and all registers with a number less than operand 3 (r_3), are stored.

- The contents of all registers used remain unchanged.

- Operand 2 (s_2) must be on a full-word boundary.

STR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
STR	03	RR	2

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Controls internal timer register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	STR	r_1, r_2

SU*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SU	7F	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	■ SIGNIFICANCE
☐ DECIMAL DIVIDE	■ SPECIFICATION:
☐ DECIMAL OVERFLOW	■ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
■ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	■ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	☐ NONE

Function:

Causes the contents of a full word in main storage specified by operand 2 to be algebraically subtracted from the contents of a full word in the register specified by operand 1(r_1). The difference is placed in a full word in the operand 1 (r_1) register.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SU	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SU	$r_1,s_2(x_2)$

Operational Consideration:

- The execution of the SU instruction is identical to that of the AU instruction, except that the sign is reversed before addition.

* *SU is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SUR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SUR	**3F**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- □ IF OVERFLOW, SET TO 3
- □ UNCHANGED

Possible Program Exceptions

- □ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- ■ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- ■ OPERATION
- □ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes the contents of a full word in the operand 2 (r_2) register to be algebraically subtracted from a full word in the operand 1 (r_1) register. The difference is placed in a full word in the operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SUR	r_1,r_2

Operational Considerations:

- The execution of the SUR instruction is identical to that of the AUR instruction, except that the sign is reversed before addition.

* *SUR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SVC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SVC	0A	RR	2

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- ■ IF OVERFLOW, SET TO 3

SEE OPER. CONSIDERATIONS

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Causes the interrupt code field (bits 24 through 31) of the current program status word (PSW) to be changed according to the contents of operand 1, a byte of immediate data in the instruction.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SVC	i_1

Operational Considerations:

- A supervisor call interrupt request is generated.
- When the interrupt is granted, the contents of operand 1 (i_1) are stored as the interrupt code (bits 24 through 31) in the current program status word (PSW). The current PSW is stored in the supervisor call old PSW location, and the contents of the supervisor call new PSW location replace the current PSW.
- The condition code is set equal to bits 34 and 35 of the supervisor call new PSW. It remains unchanged in the old PSW.

SW*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SW	6F	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT <0, SET TO 1
- ■ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ■ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ■ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of a double word in main storage specified by operand 2 to be algebraically subtracted from the contents of the double word in the register specified by operand 1 (r_1). The difference is placed in the double word operand 1 (r_1) register.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SW	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	SW	$r_1,s_2(x_2)$

Operational Consideration:

- The execution of the SW instruction is identical to that of the AW instruction, except that the sign is reversed before addition.

* *SW is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

SWR*

Floating Point

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
SWR	**2F**	**RR**	**2**

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ■ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ■ OPERATION
- ☐ PROTECTION
- ■ SIGNIFICANCE
- ■ SPECIFICATION:
  - ■ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of the double word in the operand 2 (r_2) register to be algebraically subtracted from the double word contents of the operand 1 (r_1) register. The difference is placed in the double operand 1 (r_1) register.

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	SWR	r_1,r_2

Operational Consideration:

- The execution of the SWR instruction is identical to that of the AWR instruction, except that the sign is reversed before addition.

* *SWR is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

TM

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
TM	91	SI	4

Condition Codes

- ■ SET TO 0
- ■ SET TO 1
- □ SET TO 2
- ■ SET TO 3
- SEE OPER. CONSIDERATIONS

Possible Program Exceptions

- ■ ADDRESSING
- □ DATA (INVALID SIGN/DIGIT)
- □ DECIMAL DIVIDE
- □ DECIMAL OVERFLOW
- □ EXECUTE
- □ EXPONENT OVERFLOW
- □ EXPONENT UNDERFLOW
- □ FIXED-POINT DIVIDE
- □ FIXED-POINT OVERFLOW
- □ FLOATING-POINT DIVIDE
- □ OPERATION
- ■ PROTECTION
- □ SIGNIFICANCE
- □ SPECIFICATION:
  - □ NOT A FLOATING-POINT REGISTER
  - □ OP 1 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON HALF-WORD BOUNDARY
  - □ OP 2 NOT ON FULL-WORD BOUNDARY
  - □ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - □ OP 1 NOT EVEN NUMBERED REGISTER
  - □ OP 1 NOT ODD NUMBERED REGISTER
- □ NONE

Function:

Causes one byte in main storage specified by operand 1 to be tested for 1 bits according to the 8-bit mask specified in operand 2. The condition code is set to reflect the results of the test.

Explicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	TM	$d_1(b_1),i_2$

Implicit Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	TM	s_1,i_2

Operational Considerations:

- The 1 bits of the immediate operand 2 are used to test the bits of operand 1.
- The contents of operand 1 remain unchanged.
- The condition code is set:
  - to zero if all the 1 bits in the mask match zero bits in the byte tested or if all the bits in the mask are zero;
  - to 1 if some of the 1 bits in the mask match zero bits in the byte tested; or
  - to 3 if all the 1 bits in the mask correspond with 1 bits in the byte tested.

  Code 2 is not used.

TR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
TR	DC	SS	6

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT <0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes the contents of operand 1 to be translated according to a table in main storage specified by operand 2. As a result, operand 1 will contain data copied from the operand 2 table.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	TR	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	TR	$s_1(l),s_2$

Operational Considerations:

- The 8-bit code of each character of operand 1 is used as an index to the base table address specified by operand 2. The character code located at this address 8-bit code value of operand 1 plus $d_2(b_2)$ is transferred from the table to the character position of operand 1. Thus, the original 8-bit code of operand 1 is replaced.
- Translation continues until all characters specified by the length (l) have been translated.
- The contents of the table are not changed unless overlap occurs.

- If the number of bytes to be translated is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.
- The programmer may place whatever values are required into the 256-byte translate table. When it is known what kind of bit configurations are expected as input (each unique configuration produces an address pointing to a unique table address), the desired value may be placed in the table to produce a translation.

TRT

General				Possible Program Exceptions	
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)	■ ADDRESSING	■ PROTECTION
MNEM.	HEX.			□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
TRT	DD	SS	6	□ DECIMAL DIVIDE	□ SPECIFICATION:
				□ DECIMAL OVERFLOW	□ NOT A FLOATING-POINT REGISTER
				□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
				□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
				□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
				□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
				□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
				□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
				□ OPERATION	□ NONE

Condition Codes
■ SET TO 0
■ SET TO 1
■ SET TO 2
□ SET TO 3
SEE OPER. CONSIDERATIONS

Function:

Causes the contents of operand 1 to be translated according to a table in main storage specified by operand 2. The resultant data in the table will be tested and condition code set.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	TRT	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	TRT	$s_1(l),s_2$

Operational Considerations:

- The *translate and test* (TRT) instruction searches the table in the same manner as the *translate* (TR) instruction.
- The selected byte (result byte) in the translate table is examined and tested for an all zero pattern. If the result byte is all zeros, it is ignored and the translate operation is continued. If the result byte is nonzero, the address of the corresponding operand 1 byte is stored in the least significant 24 bit positions of general register 1, the result byte is stored in the least significant 8-bit positions of general register 2, and the operation is terminated.

TRT

- The contents of both operands remain unchanged.
- If the maximum number of bytes to be translated is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.
- The condition code is set:
  - to zero if all result bytes are zero;
  - to 1 if the result byte corresponding to any except the last operand 1 byte is nonzero; or
  - to 2 if the result byte corresponding to the last operand 1 byte is nonzero.

  Code 3 is not used.

TS*

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
TS	93	SI	4

Condition Codes
■ SET TO 0
■ SET TO 1
□ SET TO 2
□ SET TO 3
SEE OPER. CONSIDERATIONS

Possible Program Exceptions	
■ ADDRESSING	■ PROTECTION
□ DATA (INVALID SIGN/DIGIT)	□ SIGNIFICANCE
□ DECIMAL DIVIDE	□ SPECIFICATION:
□ DECIMAL OVERFLOW	□ NOT A FLOATING-POINT REGISTER
□ EXECUTE	□ OP 1 NOT ON HALF-WORD BOUNDARY
□ EXPONENT OVERFLOW	□ OP 2 NOT ON HALF-WORD BOUNDARY
□ EXPONENT UNDERFLOW	□ OP 2 NOT ON FULL-WORD BOUNDARY
□ FIXED-POINT DIVIDE	□ OP 2 NOT ON DOUBLE-WORD BOUNDARY
□ FIXED-POINT OVERFLOW	□ OP 1 NOT EVEN NUMBERED REGISTER
□ FLOATING-POINT DIVIDE	□ OP 1 NOT ODD NUMBERED REGISTER
■ OPERATION	□ NONE

Function:

Causes the operand, a byte in main storage, to be read and bit position 0 to be tested. After the byte is tested and the condition code is set, all the bits in this indicator byte are set to 1. The byte indicated by the operand can be used as an indicator switch which is tested and set to all binary 1's by this instruction and then reset to binary 0's by some other instruction.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	TS	$d_1(b_1)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	TS	s_1

Operational Considerations:

- Only the first bit of the operand is tested to determine the condition code.
- All eight bits of the operand are set to binary 1's after the condition code is set.
- The condition code is set as follows:
  - 0 if bit position 0 is zero; or
  - 1 if bit position 0 is one.

* *TS is a featured instruction. If you attempt to issue this instruction to a processor which does not have the control feature installed, you cause an operation program exception.*

UNPK

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
UNPK	F3	SS	6

Condition Codes

- ☐ IF RESULT = 0, SET TO 0
- ☐ IF RESULT < 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ■ UNCHANGED

Possible Program Exceptions

■ ADDRESSING	■ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Converts the contents of operand 2 from a packed format to an unpacked format, which is placed in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	UNPK	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	UNPK	$s_1(l_1),s_2(l_2)$

Operational Consideration:

- This instruction proceeds one byte at a time from right to left. The first byte operated on has its sign and digit reversed (a 4C would become C4). Each half byte from then on is moved to the next left digit field, and an F is placed in the zone field of the receiving byte (EBCDIC notation).
- Any unfilled bytes that are part of the specified length for operand 1 are zero-filled.
- Operand 2 data should be in packed decimal format.
- Operand 1 should contain enough bytes to receive all digits, a zone for each digit, and a sign from operand 2.
- Specification of a length attribute for operands 1 and 2 is optional.

X

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
X	57	RX	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT ≠ 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ■ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ■ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes a logical exclusive OR operation to be performed on the contents of the operand 1 (r_1) register and the full word in main storage specified by operand 2. The result is placed in operand 1 (r_1).

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	X	$r_1,d_2(x_2,b_2)$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	X	$r_1,s_2(x_2)$

Operational Considerations:

- A bit position in the result is set to 1 if the corresponding bit positions in the operands are unlike; otherwise, the bit position in the result is set to zero.
- The rules of operation for the exclusive OR (X) operation are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
1	0	1
0	1	1
1	1	0

XC

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
XC	D7	SS	6

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT ≠ 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED POINT DIVIDE
- ☐ FIXED POINT OVERFLOW
- ☐ FLOATING POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING POINT REGISTER
  - ☐ OP 1 NOT ON HALF WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes a logical exclusive OR operation to be performed on the contents of the areas in main storage specified by operand 1 and operand 2. The result is placed in operand 1.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	XC	$d_1(l,b_1),d_2(b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	XC	$s_1(l),s_2$

Operational Considerations:

- A bit position in the result is set to 1 if the corresponding bit positions in the operands are unlike; otherwise, the bit position in the result is set to zero.
- The rules of operation for the exclusive OR operation are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
1	0	1
0	1	1
1	1	0

XC

- The number of bytes used in each operand is specified by l in operand 1.
- If the number of bytes to be used in each operand is not explicitly shown in operand 1, then the number will be equal to the length attribute of operand 1.

XI

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
XI	97	SI	4

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT ≠0, SET TO 1
- ☐ IF RESULT >0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ■ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ■ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ☐ NONE

Function:

Causes a logical exclusive OR operation to be performed on the contents of operand 1 (a byte in main storage) and operand 2 (a byte of immediate data in the instruction). The result is placed in operand 1.

Explicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	XI	$d_1(b_1),i_2$

Implicit Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	XI	s_1,i_2

Operational Considerations:

- A bit position in the result is set to 1 if the corresponding bit positions in the operands are unlike; otherwise, the bit position in the result is set to zero.
- The rules of operation for the exclusive OR (XI) operation are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
1	0	1
0	1	1
1	1	0

XR

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
XR	17	RR	2

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT ≠ 0, SET TO 1
- ☐ IF RESULT > 0, SET TO 2
- ☐ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

- ☐ ADDRESSING
- ☐ DATA (INVALID SIGN/DIGIT)
- ☐ DECIMAL DIVIDE
- ☐ DECIMAL OVERFLOW
- ☐ EXECUTE
- ☐ EXPONENT OVERFLOW
- ☐ EXPONENT UNDERFLOW
- ☐ FIXED-POINT DIVIDE
- ☐ FIXED-POINT OVERFLOW
- ☐ FLOATING-POINT DIVIDE
- ☐ OPERATION
- ☐ PROTECTION
- ☐ SIGNIFICANCE
- ☐ SPECIFICATION:
  - ☐ NOT A FLOATING-POINT REGISTER
  - ☐ OP 1 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON HALF-WORD BOUNDARY
  - ☐ OP 2 NOT ON FULL-WORD BOUNDARY
  - ☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
  - ☐ OP 1 NOT EVEN NUMBERED REGISTER
  - ☐ OP 1 NOT ODD NUMBERED REGISTER
- ■ NONE

Function:

Causes a logical exclusive OR operation to be performed on the contents of the registers specified by operand 1 (r_1) and operand 2 (r_2). The result is placed in operand 1 (r_1).

Explicit and Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	XR	r_1,r_2

Operational Considerations:

- A bit position in the result is set to 1 if the corresponding bit positions in the operands are unlike; otherwise, the bit position in the result is set to zero.

- The rules of operation for the exclusive OR (XR) operation are illustrated by the following truth table:

Operand 1	Operand 2	Result (Operand 1)
0	0	0
1	0	1
0	1	1
1	1	0

ZAP

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		
ZAP	F8	SS	6

Condition Codes

- ■ IF RESULT = 0, SET TO 0
- ■ IF RESULT < 0, SET TO 1
- ■ IF RESULT > 0, SET TO 2
- ■ IF OVERFLOW, SET TO 3
- ☐ UNCHANGED

Possible Program Exceptions

■ ADDRESSING	■ PROTECTION
■ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
■ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Function:

Clears operand 1 to zeros and adds the value of operand 2. Replaces operand 1 with the value of operand 2.

Explicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	ZAP	$d_1(l_1,b_1),d_2(l_2,b_2)$

Implicit Format:

LABEL	△OPERATION△	OPERAND
[symbol]	ZAP	$s_1(l_1),s_2(l_2)$

Operational Considerations:

- Equivalent to AP with zero in operand 1. Sign digit is generated.
- Checks operand 2 sign and digits for validity.
- Decimal overflow condition exists when operand 2 value will not fit in operand 1. Most significant digits are truncated.
- Zero result has positive sign. When overflow occurs, zero result has sign of operand 2.
- Operand 2 is zero extended when it does not fill operand 1.
- Operands 1 and 2 may overlap if least significant bytes coincide, or if least significant byte of operand 1 is to the right of the least significant byte of operand 2.

3. BAL Directives

CCW

Function:

Defines and generates an 8-byte *channel command word* aligned on a double-word boundary.

For full information on the use of the CCW, refer to the processor programmer reference, UP-8052 (current version).

Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CCW	op_1,op_2,op_3,op_4

where:

op_1
Is the command code specifying the operation to be performed.

op_2
Is the address of the first byte in main storage of the data being controlled.

op_3
Is the flag control indicating the options desired.

op_4
Is the byte count indicating the number of bytes of data to be controlled.

CNOP

Function:

Adjusts the location counter to a half-word, full-word, or double-word storage boundary without initiating any other operation.

Format:

LABEL	△OPERATION△	OPERAND
unused	CNOP	a_1,a_2

where:

a_1 and a_2
Are absolute expressions consisting of predefined terms.

Operational Considerations:

The first expression in the operand field indicates a byte to which the location counter must be set. Legal values for the first expression are zero and 2 for full-word boundary alignment, and zero, 2, 4, and 6 for double-word boundary alignment, as follows:

- Zero indicates a full-word or double-word boundary.
- A 2 indicates the second byte (first half word) past the boundary.
- A 4 indicates the fourth byte (second half word) past a double-word boundary.
- A 6 indicates the sixth byte (third half word) past a double-word boundary.

Permissible values for the second expression are 4 and 8, indicating that the adjustment is relative to a full-word or double-word boundary, respectively.

If the location counter is already set to the indicated byte, the CNOP has no effect. When alignment is needed, one, two, or three no-operation instructions are generated to increment the location counter to the proper half-word boundary and to ensure correct instruction processing. All terms must be predefined.

COM

Function:

Enables the programmer to define a control section to be used as a common storage area for two or more separately assembled routines. The format of the common section may be described by DS and DC directives. Labels appearing within the sections are defined. Like a dummy control section, no data or instructions are assembled in a common section. It has a separate location counter with an initial value of zero. Data may be entered into a common section only by execution of a program which refers to it. DC instructions act as DS instructions in the COM area because neither instructions nor constants in a common storage area are assembled. Labels defined in a common section are not subject to the restrictions imposed on dummy section labels.

One assembly can define only one common section. However, several COM directives may appear among the source statements. Each COM directive after the first defines a continuation of the common section previously described. When several routines defining common storage are linked, the resulting module contains only one section corresponding to the common sections in the input modules. The length of this section is the length of the largest like common section in the input modules.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	COM	unused

Operational Considerations:

If the common section is unlabeled, the area is addressed by referencing the label of a statement within the common section with a USING directive.

If more than one object module element refers to a common storage area with the same name, the references are to the same storage area. Only one common storage area is allocated within a load module to satisfy all object module requests for common storage areas with the same name. The size of a common storage area in a load module is determined by the maximum size requested by any object module for common storage with that name. Blank common storage areas are allocated in the same way.

In a multiphase load module, common storage areas are not normally overlaid.

The following rules apply to the use of common storage:

— An entry point cannot have the same name as a labeled common storage area included in the load module.

— When the linkage editor includes module elements (CSECT or COM) with the same name as a labeled common storage area, that section is treated as a block data subprogram (i.e., to initialize values of labeled common blocks) and is loaded into all or a portion of the common storage area. A block data subprogram is loaded when the phase in which it was included is loaded. Blank common cannot be initialized during loading unless the text encountered is for that COM ESD.

COM

— If an object module has requested common storage, the partial inclusion of a single control section from that object module will cause the common storage area defined to be included also, regardless of whether or not the included control section refers to that common storage name. For further information, see the linkage editor portion in system service programs (SSP) user guide, UP-8062 (current version).

COPY

Function:

Causes the source module identified in the operand field of the COPY directive to be included directly into the source program being assembled.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	COPY	symbol

where:

symbol
Identifies the code to be copied by the assembler. Only one symbol may be used.

Operational Considerations:

The assembler places the source code, identified by the operand, immediately after the COPY directive. This source module may not include any COPY, END, ICTL, MACRO, or MEND directives. Statements included in the program by a COPY directive are assumed to be in standard format regardless of any ICTL directives in the program.

CSECT

Function:

Indicates the initiation or continuation of a control section.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	CSECT	unused

Operational Considerations:

The symbolic name of the control section defines an entry point of the program being assembled. This symbol must not appear as a symbol for any other source statement except the START directive of its control section or another CSECT directive to indicate continuation of the coding in the same control section.

Each control section is adjusted to begin on a double-word boundary. The value of the symbol is the address of the first byte of the control section and has a length attribute of 1.

If the symbol is blank, the CSECT directive is a continuation of coding for an unnamed control section. If the symbol is blank and is not preceded by an unnamed control section, the CSECT initiates an unnamed control section. Only one unnamed control section is permitted in a module.

DC

Floating Point

Function:

Defines the value of a floating-point number and has a program storage location assigned to it. The format of floating-point constants differs from the standard format of the DC statement in that an additional subfield may appear — the scale modifier.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DC	[d]t[L_n][S+n]'c[E±n]'

where:

[symbol]
Is up to eight characters.

d
The duplication factor designates the number of identical constants or areas to be generated. An unsigned decimal value is used to specify the duplication factor. If no duplication subfield is used, the assembler assumes a factor of 1. A duplication factor of zero generates neither a constant nor a storage area and, if no length factor is specified, the location counter will provide the proper boundary alignment and assign the location counter value to the symbol used. A duplication factor of zero is not permitted with literals. Even though the duplication factor can change the size of the storage area used, the use of the duplication factor does not change the length attribute of the field. The maximum value of the duplication factor is 256.

t
The definition-type symbol is required to determine the alignment, padding, truncation, storage form, and implied length. (See Table A—6 for the characteristics of the E and D types.)

L_n
Is the explicit length factor in decimal. Two types of floating-point constants are available: full word (E) and double word (D). The implied length of an E type constant is four bytes; if the length modifier is omitted, full-word boundary alignment is assigned. The implied length of a D type constant is eight bytes; if the length modifier is omitted, double-word boundary alignment is assigned. In either case, an explicit length modifier of from one to eight bytes may be specified.

S+n
Is the scale modiifer and must be a positive signed or unsigned decimal number. If the sign is omitted, a positive value is assumed. The scale modifier is applied to a number after it has been converted to internal format.

DC

Floating Point

'c[E±n]'

Is the constant specification with optional exponent. A floating-point number is written as a decimal number which may be an integer (110), a fraction (75), or a mixed number (110.75). The floating-point number may be followed by an optional exponent represented by an E, a sign, and a decimal number, respectively. In the absence of a sign, a plus sign is assumed. The exponent for a constant is that power of 10 by which that constant will be multiplied before its conversion to internal format. This exponent value may range from —85 to +75.

Operational Considerations:

The machine representation of the constant consists of a hexadecimal fraction (mantissa) and a hexadecimal exponent (characteristic). The arithmetic point is assumed to be at the left of the leftmost digit of the fraction. The characteristic represents the power of 16 by which the fraction must be multiplied to obtain the value of the constant. The machine format is as follows:

(SHORT FORMAT)

FULL WORD	sign 0	characteristic (exponent) 1 — 7	mantissa (fraction) 6 hexadecimal digits 8 — 31

(LONG FORMAT)

DOUBLE WORD	sign 0	characteristic (exponent) 1 — 7	mantissa (fraction) 14 hexadecimal digits 8 — 63

where:

sign

Is the zero bit, the sign of the mantissa.

characteristic

Is a 7-bit binary number (signed and biased by the hexadecimal value 40_{16}, decimal value 64) reflecting the scaling of the floating-point number.

mantissa

Is the fraction after the constant has been converted to its machine representation; scaling is performed if specified.

NOTE:

The floating-point value is the product of the mantissa (fraction) and the base 16 raised to the power of the biased characteristic (exponent) after the exponent has been reduced by 64.

DC

Standard Format

Function:

Defines the value of a decimal number, an alphanumeric expression, or address constant and has a program storage location assigned to it.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	DC	$[d]\,t\,[L_n]\left\{\begin{matrix}'c' \\ (c)\end{matrix}\right\}$

where:

[symbol]
Is up to eight characters long.

d
The duplication factor designates the number of identical constants or areas to be generated. An unsigned decimal value is used to specify the duplication factor. If no duplication subfield is used, the assembler assumes a factor of 1. A duplication factor of zero generates neither a constant nor a storage area and, if no length factor is specified, the location counter will provide the proper boundary alignment and assigns the location counter value to the symbol used. A duplication factor of zero is not permitted with literals. Even though the duplication factor can change the size of the storage area used, the use of the duplication factor does not change the length attribute of the field. The maximum value of the duplication factor is 256.

t
The definition-type symbol is required for both DC and DS statement to determine the alignment, padding, truncation, storage form, and implied length. (See Table A—6 for the characteristics of the 13 types used.)

L_n
The length factor designates the explicit value of the length attribute of a field generated by a DS or DC statement. The length attribute of a field used in an assembler application instruction determines the number of bytes involved in that instruction. The maximum value of the length factor is 256. Boundary alignment is not provided when a length factor is specified.

'c' or (c)
The constant specification determines the constant, or storage, to be generated. When an apostrophe or ampersand is included in the constant specification, double apostrophes or ampersands are used to indicate the inclusion of these characters in the constant. The constant may take the form of data or an address, as shown in Table A—6.

DROP

Function:

Informs the assembler that the registers specified are no longer available for base register assignment.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	DROP	r_1 [,...,r_n]

where:

r_1[,...,r_n]
: Specifies that the declared registers (0 through 15) are no longer available for base register assignment.

Operational Considerations:

Registers previously made available for base register assignment may be dropped and made available again in a USING directive. The value assumed to be in a base register may be changed by coding another USING directive without an intervening drop of that register.

DS

Function:

Defines storage to be used as work areas, to hold data, and to function as input and output areas. The storage areas are assigned program locations.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	DS	$[d]\,t[L_n]\begin{bmatrix}'c'\\(c)\end{bmatrix}$

where:

symbol
Is up to eight characters long.

d
The duplication factor designates the number of identical constants or areas to be generated. An unsigned decimal value is used to specify the duplication factor. If no duplication subfield is used, the assembler assumes a factor of 1. A duplication factor of zero generates neither a constant nor a storage area and, if no length factor is specified, the location counter will provide the proper boundary alignment and assigns the location counter value to the symbol used. A duplication factor of zero is not permitted with literals. Even though the duplication factor can change the size of the storage area used, the use of the duplication factor does not change the length attribute of the field. The maximum value of the duplication factor is 256.

t
The definition-type symbol is required for both DC and DS statements to determine the alignment, padding, truncation, storage form, and implied length. (See Table A—6 for the characteristics of the 13 types used.)

L_n
The length factor designates the explicit value of the length attribute of a field generated by a DS or DC statement. The length attribute of a field used in an assembler application instruction determines the number of bytes involved in that instruction. The maximum value of the length factor is 256.

'c' or **(c)**
The constant specification determines the constant, or storage, to be generated. When an apostrophe or ampersand is included in the constant specification, double apostrophes or ampersands are used to indicate the include of these characters in the constant. The constant may take the form of data or an address, as shown in Table A—6.

NOTE:

The maximum explicit length for a DS is 65,535 bytes. (See Table A—6 for C and X types.) Only the number, not the content, of the bytes reserved by a DS statement is determined by the assembler.

DSECT

Function:

Defines a data storage area permitting one or more programs to use indirect symbolic addressing for the same record items.

Format:

LABEL	△ OPERATION △	OPERAND
[symbol]	DSECT	unused

Operational Consideration:

Storage is not reserved by a DS directive within a dummy control section, and the data and instructions appearing in a dummy control section do not become part of the assembled program. A separate location counter with an initial value of zero is kept for each dummy control section. More than one DSECT directive with the same symbol may appear in a module. The first DSECT directive initiates the dummy control section; the remaining DSECT directives continue it.

Symbols of statements in a dummy control section are called dummy section symbols. The following rules must be observed in using and assigning dummy section symbols:

- An unpaired dummy section symbol may appear only in an expression defining a storage address for a machine instruction or an S-type constant.

- A base register may not be specified for an address field containing an unpaired dummy section symbol.

- The programmer must ensure that the appropriate value is loaded into the register specified in the USING statement.

To guarantee alignment between the actual storage area and the dummy control section, the user should align the storage area to a double-word boundary.

EJECT

Function:

Causes the assembler to continue the assembly listing on the top of the next page.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	EJECT	unused

Operational Considerations:

If the next line of the listing causes a page change, the EJECT directive has no effect.

When the EJECT directive is encountered, the printing form is skipped to the next page. If a title has been previously specified, the title is printed on the new page. An EJECT directive appearing in a source code macro definition causes the form to be skipped whenever the definition is listed and each time the macro is generated.

The assembler will advance the assembly listing to a new sheet whenever a sheet is full. However, if the programmer would like each new logical part or subroutine to start at the top of a new sheet, he can use the EJECT directive whenever he wants a new sheet to start.

The EJECT directive itself is never printed.

END

Function:

Indicates the end of a source program.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
[symbol]	END	[e]

where:

e

Is a relocatable expression.

Operational Considerations:

The END directive must be the last statement in the source program. An expression in the operand field designates the point in the program where control may be transferred after the program is loaded.

ENTRY

Function:

Declares to the assembler those symbols defined by the module being assembled that may be referenced by other modules.

Format:

LABEL	△ OPERATION △	OPERAND
unused	ENTRY	symbol[,symbol,...,symbol]

Each symbol in the operand field is declared to be defined in this module. Their names and assigned values are included in the output of the assembler as external reference records.

EQU

Function:

Defines the length and value of a symbol using another symbol as all or part of the definition.

Format:

LABEL	∆ OPERATION ∆	OPERAND
symbol	EQU	e[,a]

where:

e
Is an absolute or relocatable expression.

a
Is an absolute expression.

All symbols must be predefined.

Operational Considerations:

The symbol in the label field is defined as the value of the first expression in the operand. The maximum values are -2^{23} to $2^{23}-1$. The length attribute of the symbol is equal to the second expression (a) if explicitly stated. If the second expression (a) is omitted, the symbol will have the length attribute of the first term in the first expression (e). If the first term is an * or a self-defining term, the length attribute of the symbol is 1.

EXTRN

Function:

Declares to the assembler those symbols used in the module being assembled that are defined in a different module.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	EXTRN	symbol[,symbol,...,symbol]

Operational Considerations:

Each symbol in the operand field is declared to be a symbol defined in some other module. The symbolic name and the external symbol identification assigned by the assembler are input to the linkage editor as an external definition record. Each reference to the externalized symbol creates an appropriate relocation mask to allow reference resolution at linkage editor time. When an EXTRN and a definition for an identical symbol appear in the same assembly, the EXTRN reference is discarded automatically, and the definition is accepted regardless of the order of appearance of either item.

ICTL

Function:

Specifies new values for the begin, end, and continue columns. Normally, a source statement begins in column 1 of the coding form and ends in column 71. If a continuation statement is needed, a character is written in column 72, and the statement continues in column 16 of the following line.

Format:

LABEL	△OPERATION△	OPERAND
unused	ICTL	[b] [,e] [,c]

where:

b

Is an unsigned decimal integer specifying the beginning column. It must be between 1 and 75.

e

Is a unsigned decimal integer specifying the ending column. It must be greater than or equal to b + 5.

c

Is an unsigned decimal integer specifying the continuation column. It must be greater than or equal to b and less than e. The line is continued starting in the column specified by c.

If b is omitted, it is assumed to be 1. If e is omitted, it is assumed to be 71. If c is omitted or if e equals 80, continuation records are not allowed.

Operational Considerations:

There can be only one ICTL directive in a source code module and it must immediately precede or follow any program-defined macro definitions. The ICTL directive applies only to those source statements that follow it. All library macro definitions are assumed to have normal output format. If the ICTL appears before the START card and it is incorrect, the assembly is terminated. When an ICTL appears out of sequence (must be first statement following START card) the ICTL terminates the assembly.

ISEQ

Function:

Informs the assembler which columns of the source statement contain the field used for checking the sequence of statements and controls the initiation and termination of sequence checking.

Format:

LABEL	△OPERATION△	OPERAND
unused	ISEQ	l,r

where:

l

Is a decimal integer specifying the leftmost column of the field to be used for the sequence check.

r

Is a decimal integer specifying the rightmost column of the field to be used for the sequence check; r must be greater than or equal to l.

Operational Considerations:

Columns to be checked should not fall between the beginning and ending input columns specified for the program.

The sequence check begins with the first source statement after the first ISEQ directive and is terminated by an ISEQ directive with a blank or invalid operand field.

Sequence checking is not performed on statements generated from macro definitions or on statements inserted into the source code via a COPY directive.

If no ISEQ directive is supplied, no sequence checking occurs.

LTORG

Function:

Generates all literals previously defined into a data pool within the source program.

Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	LTORG	unused

Operational Considerations:

The literals are pooled following the occurrence of the LTORG directive. A symbol in the label field represents the first byte of the generated literal pool and is assigned a length attribute of 1. LTORG directives may not appear within a dummy control section or in a blank common storage area. If there are no LTORG statements in a program and literals are specified, or if any literals are specified after the last LTORG directive in a program, these literals are pooled at the end of the first control section. The programmer then must ensure that a valid base register is available to address the locations in the literal pool.

Literals are placed in the literal pool according to their total length (duplication factor multiplied by the length of the constant). The literal pool consists of four sections:

1. Literals with total lengths that are multiples of double words (eight bytes)

2. Literals with total lengths that are multiples of full words (four bytes)

3. Literals with total lengths that are multiples of half words

4. Any remaining literals

Within each pool section, the literals are stored in order of occurrence. Before the literal pool is generated, the location counter is adjusted to a double-word boundary. If two control sections are assembled together and an LTORG is not included in the second or following sections, then all the literals defined in all the sections will be pooled in the first section and may subsequently be available only to that first section. To ensure that each linked control section can use the literals declared by it, an LTORG can be used within each control section.

OPSYM

Function:

The delete operation code (OPSYM) directive allows you to tell the assembler not to accept a certain mnemonic operation code.

Format:

LABEL	△OPERATION△	OPERAND
mnemonic operation code	OPSYM	unused

After you use the OPSYM directive to declare a mnemonic code as unacceptable, the assembler will not generate the normal object code for that mnemonic if it appears after the OPSYM. You are then free to use the declared mnemonic another way, such as the mnemonic code of a macro prototype statement.

The OPSYM directive cannot be used from within a PROC/MACRO or from within code generated as a result of conditional assembly statements.

OPSYM

Example:

```
     LABEL       ␢OPERATION␢   OPERAND
   1               10          16
1.               MACRO
2                A             &QUANT,&Q2,&SUM
3                L             1,3,&QUANT
4.               A             1,3,&Q2
5                ST            1,3,&SUM
6                MEND
7                START         0
8. A             OPSYM
                 .
                 .
                 .
9. CALCH         A             PAY,RAISE,TOTAL
                 .
                 .
                 .
10.              END
```

In this example, I preceded my program with a macro definition which I'll use in my program. Line 2 contains the mnemonic code A, which is the mnemonic operation code for an add full word instruction. Before I can call the A macro into my program, I must use an OPSYM directive to tell the assembler not to recognize A as the add full word mnemonic. The OPSYM directive must come before the line of code which references the macro; that is, line 8 must precede line 9.

ORG

Function:

Sets or resets the location counter to a specified value.

Format:

LABEL	∆OPERATION∆	OPERAND
[symbol]	ORG	[e]

where:

e

Is a relocatable expression.

Operational Considerations:

The location counter is set to the value of the expression in the operand field. When no expression is present, the location counter is set to the highest location previously assigned in that control section. A symbol in the label field has the same value as the expression in the operand field and is assigned a length attribute of 1. The expression in the operand field must be relocatable. Its value must represent an address in the same control section in which the ORG occurs. This address value must be equal to or greater than the initial setting of the current location counter. If the expression is in error, the ORG directive is ignored, and the line is flagged. All terms in the expression must be predefined.

The ORG directive permits the location counter to be set to a value not on a half-word boundary.

Bytes of storage reserved with an ORG directive are not set to zero or cleared when the program is loaded.

PRINT

Function:

Controls the contents of the assembly listing.

Format:

LABEL	△OPERATION△	OPERAND
unused	PRINT	$\left[\begin{Bmatrix}\text{ON}\\ \text{OFF}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\text{GEN}\\ \text{NOGEN}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\text{DATA}\\ \text{NODATA}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\text{SINGLE}\\ \text{DOUBLE}\end{Bmatrix}\right]$

where:

ON
Specifies the listing is to be printed.

OFF
Specifies that no listing is printed.

GEN
Specifies that lines generated by a macro instruction are printed.

NOGEN
Specifies that lines generated by a macro instruction are not printed, except that the macro instruction and any MNOTE messages generated are printed.

DATA
Specifies that all characters of each constant representation are printed.

NODATA
Specifies that only the first eight characters of each constant representation are printed.

SINGLE
Specifies that the source listing is single-spaced.

DOUBLE
Specifies that the source listing is double-spaced.

Operational Considerations:

If a PRINT directive specifies OFF plus other parameters, the other specifications are not effective until a PRINT directive is encountered that specifies the listing facility is to be turned ON. The options provided by a PRINT directive are keyword (not positional) parameters; therefore, the comma is not required if a parameter is omitted. The initial print condition of assembly printing is ON, GEN, NODATA, SINGLE. This condition remains until the first PRINT directive changes it. PRINT directives may change from only one to all of the parameters; any unspecified parameters remain in their previous condition. A PRINT directive may not appear in a macro definition.

PUNCH

Function:

Produces a record at assembly time. This directive is used to produce job control card images to precede or succeed the object module; it eliminates the necessity of manually inserting them.

Format:

LABEL	△OPERATION△	OPERAND
unused	PUNCH	'$c_1,\ldots,c_{80}$'

where:

$c_1,\ldots,c_{80}$
Represents a string of up to 80 characters produced as a record in the object code output.

Operational Considerations:

The following conditions apply to characters in the operand field.

- Up to 80 characters, including spaces, may be specified within the apostrophes.
- An apostrophe within the operand must be specified as a pair of apostrophes.
- An ampersand within the operand must be specified as a pair of ampersands.
- Spaces must be used to separate fields.
- In counting the 80 characters, a pair of ampersands or apostrophes written to express a single apostrophe, or ampersand, counts as one.

A PUNCH directive prior to the first control section of the program produces records prior to the first control section, and all others produce records after the last control section.

Variable symbol substitution is performed within the operand field.

Although the PUNCH directive may be included anywhere in the program, it may not be used before macro definitions.

REPRO

Function:

Reproduces a record in its entirety (columns 1 through 80) during assembly time. This directive is useful for producing job control card images to precede or succeed the object module and eliminates the necessity of manually inserting them.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	REPRO	unused

Operational Considerations:

This directive causes the contents of the following source record to be reproduced as a record in the assembler output. Each REPRO directive produces one record; up to 80 bytes are reproduced.

A REPRO directive prior to the first control section of the program produces records prior to the first control section, and all others produce records after the last control section.

All REPRO directives following the declaration of the first CSECT (START) produce records which appear after the object module transfer record. Although this directive may be included anywhere in the program, it cannot be used before a macro definition.

No substitution for variable symbols occurs in the record thus produced.

SPACE

Function:

Advances the paper in the printer a specified number of lines. The operand field contains an unsigned decimal integer specifying the number of lines the paper is to be advanced. If no operand is coded, one line will be spaced.

Format:

LABEL	△OPERATION△	OPERAND
unused	SPACE	[i]

where:

i

Is an unsigned decimal integer.

START

Function:

Defines the program name, the name of the first control section, and the initial location counter value.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	START	[a]

where:

a
Is an absolute expression.

Operational Considerations:

A symbol in the label field becomes the name of the first or only control section in the program. If the label field is blank, an unnamed control section is begun. All statements following the START directive are assembled as part of the control section until another unique control section definition is encountered.

The label field of a CSECT directive, which contains the same name as the label field of the START directive, identifies the continuation of the control section. A blank label field in the CSECT directive identifies the continuation of an unnamed control section that began with an unnamed START directive.

The symbol in the label field of the START directive also identifies or names the object program. If the START directive is unnamed, the object module is assigned the name ASMOBJ. The symbol must be a valid symbol. It is an automatic entry point and has a length attribute of 1. The START directive must not be preceded by any statements which would initiate a control section.

The self-defining term in the operand field of the START directive establishes the initial location counter value for the first control section. If the self-defining term represents a value which is not a multiple of 8, the START directive is flagged and the location counter set to the next higher multiple of 8. If the operand is omitted, the initial control section is assigned a location counter value of zero.

TITLE

Function:

Provides data for the heading of each page of the assembler listing and advances the printer form to a new page.

Format:

LABEL	Δ OPERATION Δ	OPERAND
unused	TITLE	'c'

where:

'c'
Is a heading of up to 100 characters enclosed in apostrophes.

Operational Considerations:

The following conditions apply to characters in the operand field:

- Any character may be specified, including spaces, within the defining apostrophes.
- An apostrophe within the operand must be specified as a pair of apostrophes.
- An ampersand within the operand must be specified as a pair of ampersands.
- Spaces may be specified freely to separate heading words.

More than one TITLE directive is permitted in a program. A TITLE directive provides the heading for all pages in the listing which succeed it.

USING

Function:

Informs the assembler that a specified register is available for base register assignment and will contain a specific value at execution time. The value must be loaded by the program into the base register that the USING directive specifies. The assembler maintains a USING table of the specified registers.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	USING	v,r_1 [$,\ldots,r_n$]

where:

v
: Is the value assumed to be in the first specified register at execution time. This value may be relocatable or absolute. Literals are not permitted.

r_1[$,\ldots,r_n$]
: Specifies that the declared registers (0 through 15) will be used as base registers loaded at execution time. These register numbers do not necessarily have to be assigned in ascending sequence.

Operational Considerations:

The first register specified after v is assigned the value of v; the next register is assigned the value of the first register plus 4096; the next register is assigned the value of the second register plus 4096; and so on through all the registers specified. A USING directive may specify a single register or a group of registers, or the registers may be specified by individual USING directives.

Register 0 may be specified as a valid base register; however, the assembler assumes that it always contains the value 0 and calculates displacement as if the operand were zero. Register 0 must be the operand specified by r_1, and any registers specified in the operand field following register 0 are assumed to contain increments of 4096 from zero.

When v is absolute, the indicated registers may be used to process only absolute effective addresses.

When v is relocatable, the indicated registers can be used to process only relocatable effective addresses. The registers $r_1,\ldots,r_n$ are used to process only those addresses in the same control section as the address represented by v.

The value specification in a USING directive sets the lower limit of an address range; the upper limit is automatically set 4095 bytes above the lower limit. The upper limit of a USING directive may be set less than 4095 bytes by being overlapped by the lower limit of another USING directive.

The range specified by a USING directive is used by the assembler to assign base register and displacement values to those effective operand addresses that fall within that range.

USING

If an operand address is specified as an effective address instead of a base register and displacement specification, the assembler searches the USING table for a value yielding a displacement of 4095 or less; if there is more than one such value, the value that yields the smallest displacement is chosen. If no value yields a valid displacement, the operand address is set to zero, and the line is flagged with an error indication. If more than one register contains the value yielding the smallest displacement, the highest numbered register is selected.

4. BAL Macro Definition Statements

ACTR

Function:

You use the ACTR statement to limit the number of AGO, AIF, GOTO, AGOB, AIFB, and DO statements that may be processed by the assembler either within a macro or within the source program.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	ACTR	SETA expression

Operational Considerations:

The ACTR statement must be written immediately following the local and global symbol declarations in either the source program or in a macro definition. There can be a separate ACTR statement in the source program and in each macro definition.

The value of the expression in the operand field may be any positive value from 1 to $2^{23}—1$. The value specified in the operand field causes a counter to be set to that value. This counter is decremented by 1 for each AGO, AGOB, or GOTO statement that is processed for each AIF or AIFB statement whose evaluation resulted in a true condition and for each time that the range of a DO statement is generated.

If prior to decrementing, the counter is zero, the following occurs. If a macro is being processed, its processing and that of any macros above it in a nest are terminated. The next statement to be processed is in the source code following the macro instruction which initiated the nest. If the source code is being processed (outside a macro definition), an END directive is generated. The assembly continues with only that portion of the program generated thus far.

If an ACTR statement is not written, the value of the counter is 4096_{16}.

AGO

Function:

Unconditionally alters the sequence of source statement processing.

Format:

LABEL	△OPERATION△	OPERAND
[$.s_1$]	{AGO, AGOB, GOTO}	$.s_2$

where:

AGO
Defines the operation.

$.s_1$
Is a sequence symbol.

$.s_2$
Is a sequence symbol defined in a following source code statement.

Operational Considerations:

The label field of the AGO statement may contain a sequence symbol. AGOB or GOTO may be used in lieu of AGO in the operation field. The sequence symbol in the operand field is the symbol of the next statement to be processed. Branching forward or backward from the AGO statement is permitted.

When an AGO statement is used in a macro definition, the sequence symbol specified in the operand field must appear in the label field of another statement in that macro definition.

AIF

Function:

Conditionally alters the sequence of source statement processing.

Format:

LABEL	△OPERATION△	OPERAND
[$.s_1$]	{ AIF / AIFB }	(b).s_2

where:

$.s_1$
Is a sequence symbol.

AIF
Defines the operation.

(b)
Is a SETB logical expression enclosed in parentheses.

$.s_2$
Is a sequence symbol defined in a source code statement.

Operational Considerations:

The label field of the AIF statement may contain a sequence symbol. AIFB is permitted in lieu of AIF in the operation code field.

Any logical expression permitted in the operand field of a SETB statement is valid in the operand field of the AIF statement except a 0 or a 1 enclosed in parentheses. The sequence symbol in the operation field must be written immediately after the parenthesis terminating the logical expression.

If, after the logical expression has been evaluated, the condition is true (a value of 1), you branch to the statement specified by the $.s_2$ portion of the operand. If the condition is false (a value of 0), the statement in the source code following the AIF statement will be the next statement to be processed. Branching either forward or backward from the AIF statement is permitted. When an AIF statement is written in a macro definition, the sequence symbol specified in the operand field must appear in the label of another statement within that macro definition.

ANOP

Function:

Enables branching. If a branch is necessary and no statement within the source code supplies the branch destination in its label field, an ANOP statement can be coded to provide a label to which to branch.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
.s	$\begin{Bmatrix} \text{ANOP} \\ \text{LABEL} \end{Bmatrix}$	unused

where:

.s
Is a sequence symbol.

ANOP
Defines the operation.

Operational Considerations:

The label field must contain a sequence symbol.

When the label field of a statement which is desired as a branch destination point already contains a symbol or variable symbol, the branch destination is indicated by preceding the statement by an ANOP statement.

LABEL is an acceptable synonym for ANOP in the operation field.

DO

Function:

Defines the starting point of the code and the numbers of times it is to be generated.

Format:

LABEL	△OPERATION△	OPERAND
[&varisymb]	DO	a

where:

&varisymb
Is an optional variable symbol.

DO
Defines the operation.

a
Is a valid SETA expression or a valid SET expression written in a macro definition in proc format.

Operational Considerations:

The expression in the operand field indicates the number of times the source code statements following the DO statement are produced in the object code. All lines of coding appearing between a DO statement and its associated ENDO statement are generated. The value of the expression in the operand field may be any value from 0 to 2^{23}—1. If the value of the expression is negative, the DO statement is flagged and ignored (that is, treated as if the value has been a 1).

The set of statements between the DO statement and its associated ENDO statement are said to be within the range of the DO statement. Any valid source code statement may be within the range of a DO statement, including other DO statements with their corresponding ENDO statements. DO statements may be nested up to 10 levels.

A variable symbol may be entered in the label field of the DO statement. When the variable symbol in the label field is specified, it is used as a counter for the number of times a set of lines within the range of a DO statement has been generated. The value of this variable symbol is 1 the first time through the set of statements; 2 the second time through; and so forth. It is referenced in the same manner as a SETA symbol.

If a DO statement is within the range of another DO statement and the nested DO statement is reentered, its count begins at 1 again. The value of the variable symbol in the label field of the DO statements is available to the statements following the ENDO statement even if the operation of the DO statement cycle is interrupted.

If an AGO, AGOB, GOTO, AIF, or AIFB statement outside the range of a DO statement results in an assembler branch to a sequence symbol inside the range of the DO statement, processing continues with the statement defining the sequence symbol. Processing proceeds from that point as though the DO statement operand had had a value of 1.

END

Function:

Signifies the end of a macro definition in PROC format.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	END	unused

Operational Considerations:

An END statement signals the end of a macro definition. The assembler pairs each END statement with the most recently encountered unpaired PROC statement. The statements between paired PROC and END statements are defined as the body of a macro definition.

ENDO

Function:

Indicates the end of the range of a DO statement.

Format:

LABEL	△ OPERATION △	OPERAND
unused	ENDO	unused

Operational Considerations:

DO and ENDO statements must be paired. For every DO statement, there must be an ENDO statement to define the end of the range.

GBL
GBLA
GBLB
GBLC

Function:

Declares global set symbols. The declarative chosen determines the range of values to which the set symbol may be set and the type of SET statement used to assign the values.

Global set symbols are initialized only once and are used to pass values back and forth between macro definitions. A global set symbol declared at the source code level is available to all macro definitions in which it is also declared.

Format:

LABEL	△OPERATION△	OPERAND
unused	$\left\{\begin{array}{l}\text{GBL}\\ \text{GBLA}\\ \text{GBLB}\\ \text{GBLC}\end{array}\right\}$	s_1 [,s_2,...,s_n]

where:

GBL
Declares a general-purpose global set symbol.

GBLA
Declares an arithmetic global set symbol.

GBLB
Declares a Boolean global set symbol.

GBLC
Declares a character global set symbol.

s_1,s_2,...,s_n
Are set symbol names.

Operational Considerations:

The operand field of the global set declaration may contain one or more set symbols. A global set symbol is considered defined when declared. It is initialized only once; that is, the first time it is declared. With subsequent declarations in other contexts, the global set symbol is available for use but is not reinitialized. A set symbol must be declared before it is available for use. A set symbol declared by a GBLA or GBLB statement is assigned an initial value of zero. A set symbol declared by a GBLC or GBL statement is assigned an initial value of a null character string.

If a set symbol is declared as a global set symbol in more than one macro definition, it must be declared with the same statement code in each macro definition.

LCL
LCLA
LCLB
LCLC

Function:

Declares local set symbols. The declarative chosen determines the values to which the set symbol may be set and the type of SET statement used to assign the values. A local set symbol is available for use only in the macro definition in which it is declared.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	{LCL, LCLA, LCLB, LCLC}	s_1 [,s_2,...,s_n]

where:

LCL
Declares a general-purpose local set symbol.

LCLA
Declares an arithmetic local set symbol.

LCLB
Declares a Boolean local set symbol.

LCLC
Declares a character local set symbol.

s_1,s_2,...,s_n
Are set symbol names.

Operational Considerations:

The operand field of the local set declaration may contain one or more set symbol names. A local set symbol is considered defined when declared. A set symbol declared by an LCLA or LCLB statement is assigned an initial value of zero.

A set symbol declared by an LCLC or LCL statement is assigned an initial value of a null character string.

MACRO

Function:

Designates the start of a macro definition written in macro format.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	MACRO	unused

Operational Considerations:

This statement may be used only in macro definitions written in macro format.

A macro definition written in macro format consists of the following elements in the order specified:

1. MACRO statement (heading)
2. Prototype statement (macro instruction format)
3. Model statements (optional)
4. MEND statement (trailer)

Macro Call Instruction

Function:

Causes a precoded set of assembler instructions (a macro definition) to be inserted into a source program at the point where the macro call instruction is located. The macro definition that is inserted into the source program is identified in the operation field of the macro call instruction.

Format:

LABEL	△ OPERATION △	OPERAND
[symbol]	call-name	$[p_1, p_2, \ldots, p_{252}]$

If a symbol appears in the label field of a macro instruction, it must be explicitly defined in the corresponding macro definition.

The operation field of the macro call instruction contains a symbol which is the name of a macro definition stored in a library or being assembled with the program source code. The operation field calls the desired macro definition. The operand field may contain from 0 through 252 operands separated by commas. Each operand of the macro call instruction is either a positional or keyword parameter that specifies a value which is passed to the corresponding symbolic parameter references in the macro definition.

The value of a positional parameter is identified by the position it holds in the operand field. Given a macro definition which expects four positional parameters to be specified, the operand field of the macro call instruction normally has the form:

p_1,p_2,p_3,p_4

An omitted operand must be indicated by writing both commas that separated it from the string.

If the second and third operands are omitted, the form of the operand field of the macro call instruction is:

$p_1,,,p_4$

If the final parameters are the ones to be omitted, the commas following the last operand specified may be dropped. If the macro definition were to be called by using only the second of four parameters, the operand field of the macro call instruction has the form:

$,p_2$

Macro Call Instruction

A macro definition may specify that some or all of its parameters are keyword parameters. The specification of a keyword parameter consists of the keyword followed by an equal sign, followed by the value being specified for the parameter. Keyword parameters are separated by commas and may be specified in any order. Consecutive commas are not required to indicate omission of a keyword parameter specification. Keyword parameters have the form:

$a=b_1,c=d_2,e=f_3$

or

$c=d_2,a=b_1,e=f_3$

A macro definition having both positional and keyword parameters is called a mixed-mode macro definition. The operand field of a mixed-mode macro instruction must contain any positional parameter specifications followed by the keyword parameter specifications being supplied. The last positional parameter specified is followed by a comma followed by the first keyword parameter specification. Mixed-mode parameters have the form:

$p_1,p_2,p_3,p_4,a=b_1,c=d_2,e=f_3$

Operational Considerations:

Each of the macro call instruction operands consists of 1 to 127 characters, with the character string satisfying the following conditions:

- May include one or more sequences of characters enclosed in single apostrophes. The apostrophes enclosing each character sequence are paired. Paired apostrophes may appear within paired apostrophes.

- May include a single apostrophe outside paired apostrophes if written as part of the following sequence: any special character except an ampersand, the letter L, an apostrophe, and a letter.

- May include an ampersand as the first character of a variable symbol if the ampersand is a single ampersand or the last ampersand of a string containing an odd number of ampersands.

- May include paired parentheses outside paired apostrophes. To determine pairing, a left parenthesis is paired with the immediately following right parenthesis (that is, no parentheses between them). Additional pairs are determined by ignoring the first pair and reapplying the rule.

- May include an equal sign only as the first character of an operand or within paired parentheses or paired apostrophes.

- May include a comma as a character in a string if the comma is enclosed in paired parentheses or paired apostrophes. A comma standing alone is interpreted as the end of an operand.

- May include a blank within paired apostrophes. A blank not enclosed in apostrophes terminates the operand field.

NOTE:

Operands can be coded on more than one line through the use of a continuation character in column 72. If a line is to be continued, the last operand on that line must be followed by a comma. A warning message is issued if a comma is not included.

MEND

Function:

Signifies the end of a macro definition written in macro format.

Format:

LABEL	△OPERATION△	OPERAND
unused	MEND	unused

Operational Considerations:

This statement is allowed only once in each macro definition, and it must be the last statement of the definition.

MEXIT

Function:

Indicates to the assembler that thy processing of a macro definition should be terminated before ending normally with a MEND statement. This statement is used when it is necessary to process only one section or operation of a macro definition rather than the entire macro definition.

Format:

LABEL	△ OPERATION △	OPERAND
unused	MEXIT	unused

Operational Considerations:

When MEXIT is encountered, the assembler terminates processing the macro definition and processes the statement in the source program following the macro call instruction that called the macro definition containing the MEXIT.

A second macro instruction with different operands may request the processing of different portions of the macro definition containing the MEXIT.

MNOTE

Function:

Generates an error message, which indicates how dangerous an error is, or to generate a comment, which supplies information. An MNOTE statement is used in a macro definition or in source code statements.

Format:

LABEL	△OPERATION△	OPERAND
unused	MNOTE	$\left\{\begin{array}{l}\text{'m'}\\ \triangle,\text{'m'}\\ \text{S,'m'}\\ *,\text{'m'}\end{array}\right\}$

In this format, you can specify: a message enclosed in apostrophes, a comma followed by a message enclosed in apostrophes, a severity code followed by a message, or an asterisk followed by a message. In all cases, the message is printed in the assembly listing source code. The severity code indicates the danger of the error which occurred. The severity code is a decimal value of 0 to 255. If you want to indicate a severity code of 1, you leave a blank space (△) followed by the error message, enclosed in apostrophes. An asterisk used as the severity code indicates that the message following it is informational and not an error. As mentioned before, any of these specifications causes the message to be printed in the assembly listing. Also, MNOTE lines are flagged as errors and listed in the diagnostics portion of the assembly listing if they don't have an asterisk in operand 1. Messages which are preceded by an asterisk are not flagged or listed in the diagnostics because they are not errors.

Variable symbols can be used as operands in an MNOTE statement.

Model Statement

Function:

Model statements are between the NAME and END statements in a proc and between the prototype and MEND statements in a macro. The model statements define the pattern of operations to be performed at assembly. Model statements do not generate object code.

Format:

LABEL	△OPERATION△	OPERAND
$\left[\begin{Bmatrix}\text{variable symbol}\\\text{sequence symbol}\\\text{symbol}\end{Bmatrix}\right]$	mnemonic code	operands

Operational Considerations:

The label field cannot contain an asterisk.

The operation field can contain the mnemonic operation code of an assembler instruction, directive, or macro definition. The field can also contain a variable symbol if you want to generate a different operation each time the macro is called. The variable symbol is restricted to seven characters, preceded by an ampersand. The operation field cannot contain the mnemonic codes END, ICTL, ISEQ, or PRINT.

The operand field can contain symbols or variable symbols. The size of the field, after the variable values are substituted, is up to 240 characters.

NAME

Function:

Supplies the mnemonic operation code by which a macro definition in proc format is referenced. The label field of this statement supplies the name of the macro definition in which it appears.

Format:

LABEL	△OPERATION△	OPERAND
call-name	NAME	pos-0

The call-name symbol in the label field of the NAME statement identifies the mnemonic operation code by which the macro definition may be referenced. This symbol must be unique; it may not be the same as the mnemonic operation code of a machine, assembler directive, or assembler instruction or duplicate the mnemonic operation code associated with any other macro definition in the source program.

In the operand field, pos-0 can be a decimal or alphanumeric value but it cannot be a variable symbol. The value in the operand field of the NAME statement is referenced as positional parameter 0 by using the same symbolic parameter you indicated in operand 1 of the PROC statement. You can vary the value for positional parameter 0 by using multiple NAME statements.

Operational Considerations:

At least one NAME statement is required for each macro definition, but more than one may be written. Each NAME statement specifies a different name (symbol) by which the macro definition may be referenced. The NAME statement must be written immediately after the PROC statement. When more than one NAME statement follows the PROC statement, only the operand of the NAME statement containing the symbol used to reference the macro definition is available to the body of the definition.

Multiple NAME statements allow the programmer to specify a different parameter for each NAME statement and to select the parameter by referencing that particular NAME statement.

PNOTE

Function:

Generates an error message or a comment. A PNOTE statement is used in a macro definition or a source code statement.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
unused	PNOTE	$\left\{\begin{matrix} * \\ \text{'e'} \end{matrix}\right\}$,'m'

In this format, there are two operand fields. In the first field, you can specify an asterisk to indicate that the message is informational and not an error, or you can specify a character expression containing up to six characters. The second operand field contains the message. It can contain up to 79 characters. Regardless of the choice you make for the first operand, the message is printed in the assembly listing source code. If it does not contain an asterisk as operand 1, a PNOTE statement is flagged as an error, and listed in the diagnostics portion of the assembly listing. If there is an asterisk in the first operand field, the line is not flagged or listed in diagnostics. This is done because asterisk indicates that the message is not an error.

Variable symbols can be used as operands in a PNOTE statement.

PROC

Function:

Designates the start of a macro definition written in proc format.

Format:

LABEL	△OPERATION△	OPERAND
[&symbol]	PROC	[&pos,n] [,&key$_1$=,...,&key$_m$=]

where:

&symbol
Is a variable symbol referring to the label of the macro instruction.

&pos,n
Is a variable symbol used in the body of the PROC to reference positional parameters in the call instruction. The n is a decimal number indicating how many positional parameters there are.

&key$_1$=,...,&key$_m$ =
Specifies the keyword parameters. (If only keyword parameters are specified, commas must be coded in operands 1 and 2.)

Operational Considerations:

A macro definition written in proc format consists of the following elements in the order specified.

1. PROC statement (heading)
2. NAME statements
3. Model statements (optional)
4. END statement (trailer)

Macro definitions may contain either a macro or a proc format within a definition, but not both.

PROC

The functions of the PROC statement are:

- to designate the beginning of a macro definition;
- to identify the variable symbol if any, that refers to the label of the macro instruction;
- to specify the maximum number of positional parameters in the macro instruction calling a macro definition;
- to identify the variable symbols to be used to address the positional and keyword parameters in the operand field of the macro instruction; and
- to optionally specify a default value for each keyword. Values assigned to keyword parameters are set to null if nothing follows the equal sign. If a default setting is provided, the respective keyword is set to that value when the proc is called. The setting then remains unchanged if the keyword is not specified with an appropriate value on the call line.

Prototype Statement

Function:

Provides the mnemonic operation code by which a macro instruction may call a macro definition written in macro format. It names the macro definition. The prototype statement specifies the names of the positional parameters in the macro instruction that call the macro definition containing the prototype statement.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
&symbol	**call-name**	**&pos$_1$,...,&pos$_n$,&key$_1$=,...,&key$_m$=**

where:

&symbol
Is a variable symbol that refers to the symbol in the label field of the macro call instruction.

call-name
Is the symbol that is the name of the macro definition.

&pos$_1$,...,&pos$_n$
Are variable symbols used as positional parameters.

&key$_1$=,...,&key$_m$ =
Are variable symbols used as keyword parameters.

Operational Considerations:

If the label field of the prototype statement is blank, or if the variable symbol specified does not also appear in the label field of a model statement generated by the macro definition, the symbol in the label field of the macro instruction will not be defined when the macro is generated. This symbol must not duplicate the name of any parameter or set symbol defined within the prototype statement.

The operand field of the prototype statement contains the names of all the symbolic parameters wich may be coded for the macro. Zero through 252 positional and keyword parameters are permitted in the operand field. If the macro instruction contains a mixture of both positional and keyword parameters, the names of all the positional parameters must precede the names of the keyword parameters. The names of the positional parameters must appear in the order specified in the operand field of each macro call instruction.

Within the operand field of the prototype statement, the entry defining a positional parameter consists entirely of the variable symbol that names the parameter. The entry for a keyword parameter consists of the variable symbol naming the parameter followed by an equal sign. The equal sign may be optionally followed by a string of characters specifying a default value for that parameter. If no specification for the parameter is supplied in the macro call instruction, the default value is the value supplied for a reference to that parameter within a macro definition. The default value must be written following the rules for macro instruction operands. As many continuation lines may be used as required to contain the symbolic parameters and the desired comments.

SET

Function:

Assigns either an arithmetic or character string value to a variable symbol declared by an LCL or GBL statement.

Format:

LABEL	△OPERATION△	OPERAND
&s	SET	$\left\{\begin{matrix} a \\ c \end{matrix}\right\}$

where:

&s
Is a set symbol declared by LCL or GBL.

SET
Defines the operation.

a
Is a valid arithmetic expression.

c
Is a valid character expression.

Operational Considerations:

When the operand of the SET statement contains an arithmetic expression, the value of the expression may range from -2^{23} to $+2^{23}-1$. When the operand of the SET statement contains a character expression, the maximum length that may be specified is eight characters.

If a SET variable symbol is assigned a character value, a reference to the SET symbol yields the same result as a reference to SETC symbol assigned the same character value. Similarly, if a SET variable symbol is assigned an arithmetic value, a reference to the SET symbol yields the same result as a reference to a SETA symbol assigned the same value. A SET variable symbol with a character value may be reassigned an arithmetic value, and vice versa.

A SET expression is a SETA expression allowing the use of the operators >, <,=, **, and ++ in the SET expression when an arithmetic operator is valid. The characters ** represent the logical product AND, and the characters ++ represent the logical sum OR.

Each bit of the first term is compared with its corresponding bit in the second term, and the result of the comparison is placed in the corresponding position in the resulting term. The result of the bit comparison for each operator is:

AND		
A**B		Result
1	1	1
1	0	0
0	1	0
0	0	0

OR		
A++B		Result
1	1	1
1	0	1
0	1	1
0	0	0

The three relational operators are the equal (=) operator, the greater than (>) operator, and the less than (<) operator:

=

Compares the value of two terms or expressions. If the two values are equal, the assembler assigns a value of 1 to the expression. If the values are not equal, a zero value is assigned.

>

Compares two terms or expressions. If the value of the first (left) term is greater than the value of the second (right) term, a value of 1 is assigned to the expression. If the value of the second term is greater than the value of the first term, a zero value is assigned.

<

Compares the value of the first (left) expression or term with the second (right) expression or term. If the value of the first expression or term is less than the value of the second, a value of 1 is assigned to the expression. If the value of the second expression or term is less than the value of the first, a zero value is assigned.

Given the expression A+B > C, if the expression A+B has a greater value than the value of C, the assembler assigns a value of 1 to the expression. If the value of C is greater than the value of A+B, a zero value is assigned.

Since the value of a relational or logical expression is arithmetic, the expression may be used as a term in an arithmetic expression. The following chart shows operator priority.

SET

Operator	Hierarchy
*,/	5
+,—	4
**	3
++	2
<>=	1

Four statements are provided to assign values to set symbols: SETA, SETB, SETC, and SET. The statement used depends on the statement chosen to declare the set symbol. SETA, SETB, and SETC statements may be used only within macro definitions written in macro format. The SET statement may be used only within macro definitions written in proc format.

SETA

Function:

Assigns an arithmetic value to a variable symbol that was declared by an LCLA or GBLA statement.

Format:

LABEL	△OPERATION△	OPERAND
&s	SETA	a

where:

&s
Is a set symbol declared by either LCLA or GBLA.

SETA
Defines the operation.

a
Is a valid SETA term or an arithmetic combination of valid SETA terms.

Operational Considerations:

A valid SETA term is:

- a self-defining term; or
- a variable symbol with an arithmetic value; or
- a character value consisting of one to eight decimal digits.

The arithmetic operators used in writing SETA expressions are +, —, *, and /. The expression may not begin with an operator. Two operators or two terms may not succeed one another.

The rules of precedence for the evaluation of a SETA arithmetic expression are the same as stated for a SET statement. The value of a SETA expression may range from -2^{23} to $2^{23}-1$.

When the SETA symbol is used in an arithmetic expression, the arithmetic value of the symbol is substituted for the symbol. If the SETA symbol is used in another context, the arithmetic value of the SETA symbol is converted to a decimal integer with leading zeros removed. A leading minus sign will be retained. This decimal value is then substituted for the SETA symbol. If the value of the SETA symbol is zero, a single zero is substituted.

Four statements are provided to assign values to set symbols: SETA, SETB, SETC, and SET. The statement used depends on the statement chosen to declare the set symbol. SETA, SETB, and SETC statements may be used only within macro definitions written in macro format. The SET statement may be used only within macro definitions written in proc format.

SETB

Function:

Assigns a binary value of 0 or 1 to a variable symbol which was declared by an LCLB or GBLB statement.

Format:

LABEL	△OPERATION△	OPERAND
&s	SETB	b

where:

&s
Is a set symbol declared in either LCLB or GBLB.

SETB
Defines the operation.

b
Is a valid logical expression, a 0 or a 1, that must be enclosed in parentheses.

Operational Considerations:

The logical expression in the operand field may have a value of either 0 (false) or 1 (true), and the set symbol specified in the name field of the set statement is assigned the resultant binary value. The logical expression may consist of a single term or logical combination of terms.

The permissible terms are:

- a SETB arithmetic relational expression;
- a SETB character relational expression; and
- a SETB symbol.

The SETB logical operators that may be used to combine the terms are AND, OR, and NOT. The logical expression must not contain two terms in succession. Two operators may appear in succession if the first operator is either AND or OR, and the second operator is NOT. Only the operator NOT is allowed prior to the first term of the expression.

SETB

A SETB arithmetic relational expression consists of two arithmetic expressions connected by a SETB relational operator. A SETB character relational expression consists of two character strings connected by a SETB relational operator. The SETB relational operators are:

Operator	Meaning
NE	Not equal
EQ	Equal
LT	Less than
LE	Less than or equal
GT	Greater than
GE	Greater than or equal

The arithmetic expression that may be used as a term in the SETB arithmetic relational expression is defined under the SETA statement. The rules under the SETC statement define the format of the character string that may be used in a SETB character relational expression. If two character strings are of unequal length, the shorter will always compare less than the longer, regardless of actual value. The maximum length of character strings that may be compared is 127 characters.

In writing SETB expressions, the SETB relational or logical operators must be preceded and followed by at least one blank or other special character. The relational expression may be optionally enclosed in parentheses.

The procedure for evaluating a SETB expression is:

- Each term (SETB symbol, SETB arithmetic expression, or SETB character expression) is evaluated and given a value of either 1 (true) or 0 (false).

- Evaluation is from left to right. The weight of the logical operators is:

  OR = 1

  AND = 2

  XOR = 3

  Therefore, XOR is performed prior to AND, and AND is performed prior to OR.

If a SETB variable symbol is used in the operand field of a SETA or DO statement, or in an arithmetic relation (in either a SETB or AIF term), the binary values 0 and 1 are converted to the arithmetic values +0 and +1.

If the SETB variable symbol is used in the operand field of a SET statement, the value substituted is dependent on the context. In an arithmetic expression, +1 or +0 is substituted. In a character expression, the character values 1 and 0 are substituted.

SETB

Four statements are provided to assign values to set symbols: SETA, SETB, SETC, and SET. The statement used depends on the statement chosen to declare the set symbol. SETA, SETB, and SETC statements may be used only within macro definitions written in macro format. The SET statement may be used only within macro definitions written in proc format.

SETC

Function:

Assigns a character value to a variable symbol that was declared by an LCLC of GBLC statement.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
&s	SETC	c

where:

&s
Is a set symbol declared by either LCLC or GBLC.

SETC
Defines the operation.

c
Is a valid SETC operand.

Operational Considerations:

A SETC operand must be a character expression.

The maximum length of the value that may be specified for a SETC symbol is eight characters. If more than eight characters are specified, only the leftmost eight characters are used by the assembler.

Four statements are provided to assign values to set symbols: SETA, SETB, SETC, and SET. The statement used depends on the statement chosen to declare the set symbol. SETA, SETB, and SETC statements may be used only within macro definitions written in macro format. The SET statement may be used only within macro definitions written in proc format.

Appendix A. Assembler References

Table A—1. Instruction Formats (Part 1 of 2)

Instruction Type	Source Code Instruction Format		Object Code Instruction Format						
	Explicit Form	**Implicit Form**	First Half Word: Byte 1 (0–7)	Byte 2 (8–11)	Byte 2 (12–15)	Second Half Word, Bytes 3 and 4 (16–19)	(20–31)	Third Half Word, Bytes 5 and 6 (32–35)	(36–47)
RR	[symbol] opcode r_1,r_2 1	[symbol] opcode r_1,r_2		reg op 1	reg op 2				
			opcode	r_1	r_2				
RX	[symbol] opcode $r_1,d_2(x_2,b_2)$ 2	[symbol] opcode $r_1,s_2(x_2)$		reg op 1	address operand 2				
			opcode	r_1	x_2	b_2	d_2		
RS	[symbol] opcode $r_1,r_3,d_2(b_2)$ 3	[symbol] opcode r_1,r_3,s_2		reg op 1	reg op 3	address operand 2			
			opcode	r_1	r_3	b_2	d_2		
SI	[symbol] opcode $d_1(b_1),i_2$ 4	[symbol] opcode s_1,i_2		immediate operand		address operand 1			
			opcode	i_2		b_1	d_1		
SS	[symbol] opcode $d_1(l,b_1),d_2(b_2)$	[symbol] opcode $s_1(l),s_2$		length op 1 and op 2		address operand 1		address operand 2	
			opcode	l−1		b_1	d_1	b_2	d_2
SS	[symbol] opcode $d_1(l_1,b_1),d_2(l_2,b_2)$	[symbol] opcode $s_1(l_1),s_2(l_2)$		length op 1	length op 2	address operand 1		address operand 2	
			opcode	l_1-1	l_2-1	b_1	d_1	b_2	d_2

NOTES:

1 The RR instruction has three other forms:

[symbol] opcode i_1 for the SVC instruction;

[symbol] opcode r_1 for the SPM instruction; and

[symbol] opcode m_1, r_2 for the BCR instruction.

2 The RX instruction BC is written in the form:

[symbol] opcode $m_1, d_2(x_2, b_2)$

3 The RS shift instructions are written without use of the r_3 operand, in the form:

[symbol] opcode $r_1,d_2(b_2)$

4 Some SI instructions, such as TS, SSM, and SIO, do not use an i_2 field. They are written in the form:

[symbol] opcode $d_1(b_1)$

Table A—1. Instruction Formats (Part 2 of 2)

Characters	Meaning
OPCODE	The application instruction operation code.
r_1	The number of the general register containing operand 1
r_2	The number of the general register containing operand 2
r_3	The number of the general register containing operand 3
x_2	The number of the general register containing an index number for operand 2 of the RX instruction
i_1	The immediate data used as operand 1 of the SVC instruction
i_2	The immediate data used as operand 2 of an SI instruction
l	The length of the operands as stated in source code*
l_1	The length of operand 1 as stated in source code*
l_2	The length of operand 2 as stated in source code*
b_1	The number of the general register containing the base address for operand 1
b_2	The number of the general register containing the base address for operand 2
d_1	The displacement for the base address of operand 1
d_2	The displacement for the base address of operand 2
m_1	The mask used as operand 1
op_1	Operand 1
op_2	Operand 2
op_3	Operand 3
s_1	The symbol used to identify operand 1 in the implicit format
s_2	The symbol used to identify operand 2 in the implicit format

*This is coded as the true source code length of the operand, not the length less 1, as assembled in the object code. The assembler makes a reduction of 1 in the length when converting source code to object code.

Table A—2. Instruction Repertoire (Part 1 of 16)

Listing By Mnemonic Code					
Mnemonic	Instruction Name	Machine Code	Byte Length	Source Code Format	
				Explicit	Implicit
A	Add	5A	4	$r_1,d_2(x_2,b_2)$	$r_1s_2(x_2)$
AD*	Add normalized, long	6A	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
ADR*	Add normalized, long	2A	2	r_1,r_2	r_1,r_2
AE*	Add normalized, short	7A	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
AER*	Add normalized, short	3A	2	r_1,r_2	r_1,r_2
AH	Add half word	4A	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
AI	Add immediate	9A	4	$d_1(b_1),i_2$	s_1,i_2
AL*	Add logical	5E	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
ALR*	Add logical	1E	2	r_1,r	r_1,r_2
AP	Add decimal	FA	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),s_2(l_2)$
AR	Add	1A	2	r_1,r_2	r_1,r_2
AU*	Add unnormalized, short	7E	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
AUR*	Add unnormalized, short	3E	2	r_1,r_2	r_1,r_2
AW*	Add unnormalized, long	6E	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
AWR*	Add unnormalized, long	2E	2	r_1,r_2	r_1,r_2
BAL	Branch and link	45	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
BALR	Branch and link	05	2	r_1,r_2	r_1,r_2
BAS	Branch and store	4D	4	compatibility	
BASR	Branch and store	0D	2	mode only	
BC	Branch on condition	47	4	$i,d_2(x_2,b_2)$	$i,s_2(x_2)$
BCR	Branch on condition	07	2	i,r_2	i,r_2
BCT	Branch on count	46	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
BCTR	Branch on count	06	2	r_1,r_2	r_1,r_2
BXH*	Branch on index high	86	4	$r_1,r_3,d_2(b_2)$	r_1,r_3,s_2
BXLE*	Branch on index low or equal	87	4	$r_1,r_3,d_2(b_2)$	r_1,r_3,s_2
C	Compare algebraic	59	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
CD*	Compare, long	69	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
CDR*	Compare, long	29	2	r_1,r_2	r_1,r_2
CE*	Compare, short	79	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
CER*	Compare, short	39	2	r_1,r_2	r_1,r_2
CH	Compare half word	49	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
CL	Compare logical	55	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
CLC	Compare logical	D5	6	$d_1,(l,b_1),d_2(b_2)$	$s_1(l),s_2$
CLI	Compare logical immediate	95	4	$d_1(b_1),i_2$	s_1,i_2
CLR	Compare logical	15	2	r_1,r_2	r_1,r_2
CP	Compare decimal	F9	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),s_2(l_2)$
CR	Compare algebraic	19	2	r_1,r_2	r_1,r_2

*Micro expansion feature

Table A—2. Instruction Repertoire (Part 2 of 16)

Listing By Mnemonic Code					
Mnemonic	Instruction Name	Machine Code	Byte Length	Source Code Format	
				Explicit	Implicit
CVB	Convert to binary	4F	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
CVD	Convert to decimal	4E	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
D	Divide	5D	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
DD*	Divide, long	6D	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
DDR*	Divide, long	2D	2	r_1,r_2	r_1,r_2
DE*	Divide, short	7D	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
DER*	Divide, short	3D	2	r_1,r_2	r_1,r_2
DIAG	Diagnose	83	4	(Privileged)	(Privileged)
DP	Divide decimal	FD	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),,s_2(l_2)$
DR*	Divide	1D	2	r_1,r_2	r_1,r_2
ED	Edit	DE	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
EDMK*	Edit and mark	DF	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
EX	Execute	44	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
HDR*	Halve, long	24	2	r_1,r_2	r_1,r_2
HER*	Halve, short	34	2	r_1,r_2	r_1,r_2
HPR	Halt and proceed	99	4	(Privileged)	(Privileged)
IC	Insert Character	43	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
ISK*	Insert storage key	09	2	(Privileged)	(Privileged)
L	Load	58	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
LA	Load address	41	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
LCDR*	Load complement, long	23	2	r_1,r_2	r_1,r_2
LCER*	Load complement, short	33	2	r_1,r_2	r_1,r_2
LCR*	Load complement	13	2	r_1,r_2	r_1,r_2
LCS	Load control storage	B1	4	(Privileged)	(Privileged)
LD*	Load, long	68	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
LDR*	Load, long	28	2	r_1,r_2	r_1,r_2
LE*	Load, short	78	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
LER*	Load, short	38	2	r_1,r_2	r_1,r_2
LH	Load half word	48	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
LM	Load multiple	98	4	$r_1,r_3,d_2(b_2)$	r_1,r_3,s_2
LNDR*	Load negative, long	21	2	r_1,r_2	r_1,r_2
LNER*	Load negative, short	31	2	r_1,r_2	r_1,r_2
LNR*	Load negative	11	2	r_1,r_2	r_1,r_2
LPDR*	Load positive, long	20	2	r_1,r_2	r_1,r_2
LPER*	Load positive, short	30	2	r_1,r_2	r_1,r_2
LPR*	Load positive	10	2	r_1,r_2	r_1,r_2
LPSW	Load program status word	82	4	(Privileged)	(Privileged)
LR	Load	18	2	r_1,r_2	r_1,r_2
LTDR*	Load and test, long	22	2	r_1,r_2	r_1,r_2
LTER*	Load and test, short	32	2	r_1,r_2	r_1,r_2
LTR	Load and test	12	2	r_1,r_2	r_1,r_2
M	Multiply	5C	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
MD*	Multiply, long	6C	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
MDR*	Multiply, long	2C	2	r_1,r_2	r_1,r_2

*Micro expansion feature

Table A—2. Instruction Repertoire (Part 3 of 16)

Listing By Mnemonic Code					
Mnemonic	Instruction Name	Machine Code	Byte Length	Source Code Format	
				Explicit	Implicit
ME*	Multiply, short	7C	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
MER*	Multiply, short	3C	2	r_1,r_2	r_1,r_2
MH*	Multiply half word	4C	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
MP	Multiple decimal	FC	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),s_2(l_2)$
MR*	Multiply	1C	2	r_1,r_2	r_1,r_2
MVC	Move characters	D2	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l), s_2$
MVI	Move immediate	92	4	$d_1(b_1),i_2$	s_1,i_2
MVN	Move numerics	D1	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
MVO	Move with offset	F1	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),s_2(l_2)$
MVZ	Move zones	D3	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
N	AND logical	54	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
NC	AND logical	D4	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
NI	AND logical immediate	94	4	$d_1(b_1),i_2$	s_1,i_2
NR	AND logical	14	2	r_1,r_2	r_1,r_2
O	OR logical	56	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
OC	OR logical	D6	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
OI	OR logical immediate	96	4	$d_1(b_1),i_2$	s_1,i_2
OR	OR logical	16	2	r_1,r_2	r_1,r_2
PACK	Pack	F2	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),s_2(l_2)$
S	Subtract	5B	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
SD*	Subtract normalized, long	6B	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
SDR*	Subtract normalized, long	2B	2	r_1,r_2	r_1,r_2
SE*	Subtract normalized, short	7B	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
SER*	Subtract normalized, short	3B	2	r_1,r_2	r_1,r_2
SH	Subtract half word	4B	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
SIO	Start I/O	9C	4	(Privileged)	(Privileged)
SL*	Subtract logical	5F	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
SLA*	Shift left single algebraic	8B	4	$r_1,d_2(b_2)$	r_1,s_2
SLDA*	Shift left double algebraic	8F	4	$r_1,d_2(b_2)$	r_1,s_2
SLDL*	Shift left double logical	8D	4	$r_1,d_2(b_2)$	r_1,s_2
SLL	Shift left single logical	89	4	$r_1,d_2(b_2)$	r_1,s_2
SLM	Supervisor load multiple	B8	4	(Privileged)	(Privileged)
SLR*	Subtract logical	1F	2	r_1,r_2	r_1,r_2
SP	Subtract decimal	FB	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),s_2(l_2)$
SPM	Set program mask	04	2	r_1	r_1
SR	Subtract	1B	2	r_1,r_2	r_1,r_2
SRA*	Shift right single algebraic	8A	4	$r_1,d_2(b_2)$	r_1,s_2
SRDA*	Shift right double algebraic	8E	4	$r_1,d_2(b_2)$	r_1,s_2
SRDL*	Shift right double logical	8C	4	$r_1,d_2(b_2)$	r_1,s_2
SRL	Shift right single logical	88	4	$r_1,d_2(b_2)$	r_1,s_2

*Micro expansion feature

Table A—2. Instruction Repertoire (Part 4 of 16)

Listing By Mnemonic Code					
Mnemonic	Instruction Name	Machine Code	Byte Length	Source Code Format	
				Explicit	Implicit
SSFS	SOFTSCOPE forward scan	A2	4	(Privileged)	(Privileged)
SSK*	Set system key	08	2	(Privileged)	(Privileged)
SSM	Set system mask	80	4	(Privileged)	(Privileged)
SSRS	SOFTSCOPE reverse scan	A3	4	(Privileged)	(Privileged)
SSTM	Supervisor store multiple	B0	4	(Privileged)	(Privileged)
ST	Store	50	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
STC	Store character	42	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
STD*	Store long	60	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
STE*	Store short	70	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
STH	Store half word	40	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
STM	Store multiple	90	4	$r_1,r_3,d_2(b_2)$	r_1,r_3,s_2
STR	Service timer register	03	2	(Privileged)	(Privileged)
SU*	Subtract unnormalized, short	7F	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
SUR*	Subtract unnormalized, short	3F	2	r_1,r_2	r_1,r_2
SVC	Supervisor call	0A	2	i	i
SW*	Subtract unnormalized, long	6F	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
SWR*	Subtract unnormalized, long	2F	2	r_1,r_2	r_1,r_2
TM	Test under mask	91	4	$d_1(b_1),i_2$	s_1,i_2
TR	Translate	DC	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
TRT	Translate and test	DD	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
TS*	Test and set	93	4	$d_1(b_1)$	s_1
UNPK	Unpack	F3	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),s_2(l_2)$
X	Exclusive OR	57	4	$r_1,d_2(x_2,b_2)$	$r_1,s_2(x_2)$
XC	Exclusive OR	D7	6	$d_1(l,b_1),d_2(b_2)$	$s_1(l),s_2$
XI	Exclusive OR, immediate	97	4	$d_1(b_1),i_2$	s_1,i_2
XR	Exclusive OR	17	2	r_1,r_2	r_1,r_2
ZAP	Zero and add decimal	F8	6	$d_1(l_1,b_1),d_2(l_2,b_2)$	$s_1(l_1),s_2(l_2)$

*Micro expansion feature

Table A—2. Instruction Repertoire (Part 5 of 16)

Listing By Alphabetic Instructions			
Instruction Name		**Machine Code**	**Mnemonic**
Add	(Native and 360/20 Modes)	1A	(C)AR
Add		5A	A
Add decimal		FA	(C)AP
Add half word	(Native and 360/20 Modes)	4A	(C)AH
Add half word	(9200/9300 Mode only)	(AA)	(C)AH
Add immediate		9A	AI
Add immediate	(9200/9300 Mode only)	(A6)	(C)AI
Add logical	(9200/9300 Mode only)	1E	(F)ALR
Add logical		5E	(F)AL
Add normalized (long)		2A	(F)ADR
Add normalized (long)		6A	(F)AD
Add normalized (short)		3A	(F)AER
Add normalized (short)		7A	(F)AE
Add unnormalized (long)		2E	(F)AWR
Add unnormalized (long)		6E	(F)AW
Add unnormalized (short)		3E	(F)AUR
Add unnormalized (short)		7E	(F)AU
And		14	NR
And		54	N
And		94	(C)NI
And	(Native and 9200/9300 Modes)	D4	(C)NC
Branch and link		05	BALR
Branch and link	(Native and 9200/9300 Modes)	45	(C)BAL
Branch and store	(360/20 Mode only)	4D	(C)BAS
Branch and store	(360/20 Mode only)	0D	(C)BASR
Branch on condition	(Native and 360/20 Modes)	07	(C)BCR
Branch on condition		47	(C)BC
Branch on count		06	BCTR
Branch on count		46	BCT

Table A—2. Instruction Repertoire (Part 6 of 16)

Listing By Alphabetic Instructions		
Instruction Name	**Machine Code**	**Mnemonic**
Branch on index high	86	(F)BXH
Branch on index low or equal	87	(F)BXLE
Compare	19	CR
Compare	59	C
Compare decimal	F9	(C)CP
Compare half word	49	(C)CH
Compare logical	15	CLR
Compare logical	55	CL
Compare logical	95	(C)CLI
Compare logical	D5	(C)CLC
Compare (long)	29	(F)CDR
Compare (long)	69	(F)CD
Compare (short)	39	(F)CER
Compare (short)	79	(F)CE
Convert to binary	4F	CVB
Convert to decimal	4E	CVD
Diagnose – privileged	83	DIAG
Divide	1D	(F)DR
Divide	5D	D
Divide decimal	FD	(C)DP
Divide (long)	2D	(F)DDR
Divide (long)	6D	(F)DD
Divide (short)	3D	(F)DER
Divide (short)	7D	(F)DE
Edit	DE	(C)ED
Edit and mark	DF	(F)EDMK
Exclusive OR	17	XR
Exclusive OR	57	X
Exclusive OR	97	XI

Table A—2. Instruction Repertoire (Part 7 of 16)

Listing By Alphabetic Instructions		
Instruction Name	**Machine Code**	**Mnemonic**
Exclusive OR	D7	XC
Execute	44	EX
Halt and proceed – privileged	99	HPR
Halve (long)	24	(F)HDR
Halve (short)	34	(F)HER
Insert character	43	IC
Insert storage key – privileged	09	(F)ISK
Load	18	LR
Load	58	L
Load address	41	LA
Load and test	12	LTR
Load and test (long)	22	(F)LTDR
Load and test (short)	32	(F)LTER
Load complement	13	(F)LCR
Load complement (long)	23	(F)LCDR
Load complement (short)	33	(F)LCER
Load control storage – privileged	B1	LCS
Load half word	48	(C)LH
Load (long)	28	(F)LDR
Load (long)	68	(F)LD
Load multiple	98	LM
Load negative	11	(F)LNR
Load negative (long)	21	(F)LNDR
Load negative (short)	31	(F)LNER
Load positive	10	(F)LPR
Load positive (long)	20	(F)LPDR
Load positive (short)	30	(F)LPER
Load PSW – privileged	82	LPSW
Load (short)	38	(F)LER

Table A—2. Instruction Repertoire (Part 8 of 16)

Listing By Alphabetic Instructions		
Instruction Name	**Machine Code**	**Mnemonic**
Load (short)	78	(F)LE
Move	92	(C)MVI
Move	D2	(C)MVC
Move numerics	D1	(C)MVN
Move with offset	F1	(C)MVO
Move zones (Native and 9200/9300 Modes)	D3	(C)MVZ
Multiply	1C	(F)MR
Multiply	5C	M
Multiply decimal	FC	(C)MP
Multiply half word	4C	(F)MH
Multiply (long)	2C	(F)MDR
Multiply (long)	6C	(F)MD
Multiply (short)	3C	(F)MER
Multiply (short)	7C	(F)ME
OR	16	OR
OR	56	O
OR	96	(C)OI
OR (Native and 9200/9300 Modes)	D6	(C)OC
Pack	F2	(C)PACK
Service timer register – privileged	03	STR
Set program mask	04	SPM
Set storage key – privileged	08	(F)SSK
Set system mask – privileged	80	SSM
Shift left double	8F	(F)SLDA
Shift left double logical	8D	(F)SLDL
Shift left single	8B	(F)SLA
Shift left single logical	89	SLL
Shift right double	8E	(F)SRDA
Shift right double logical	8C	(F)SRDL
Shift right single	8A	(F)SRA
Shift right single logical	88	SRL

Table A—2. Instruction Repertoire (Part 9 of 16)

Listing By Alphabetic Instructions			
Instruction Name		**Machine Code**	**Mnemonic**
SOFTSCOPE forward scan – privileged		A2	SSFS
SOFTSCOPE reverse scan – privileged		A3	SSRS
Start I/O – privileged		9C	SIO
Store		50	ST
Store character		42	STC
Store half word		40	(C)STH
Store (long)		60	(F)STD
Store multiple		90	STM
Store (short)		70	(F)STE
Subtract	(Native and 360/20 Modes)	1B	(C)SR
Subtract		5B	S
Subtract decimal		FB	(C)SP
Subtract half word	(Native and 360/20 Modes)	4B	(C)SH
Subtract half word	(9200/9300 Mode only)	(AB)	(C)SH
Subtract logical		1F	(F)SLR
Subtract logical		5F	(F)SL
Subtract normalized (long)		2B	(F)SDR
Subtract normalized (long)		6B	(F)SD
Subtract normalized (short)		3B	(F)SER
Subtract normalized (short)		7B	(F)SE
Subtract unnormalized (long)		2F	(F)SWR
Subtract unnormalized (long)		6F	(F)SW
Subtract unnormalized (short)		3F	(F)SUR
Subtract unnormalized (short)		7F	(F)SU
Supervisor call		0A	SVC
Supervisor load multiple – privileged		B8	SLM
Supervisor store multiple – privileged		B0	SSTM
Test and set		93	(F)TS
Test under mask		91	(C)TM

Table A—2. Instruction Repertoire (Part 10 of 16)

Listing By Alphabetic Instructions		
Instruction Name	**Machine Code**	**Mnemonic**
Translate	DC	(C)TR
Translate and test	DD	TRT
Unpack	F3	(C)UNPK
Zero and add	F8	(C)ZAP

NOTES:

1. Tag symbol (F) before mnemonic indicates instructions that are added as features.

2. Tag symbol (C) before mnemonic indicates instruction available in native mode and in 9200/9300 and 360/20 compatibility modes, unless indicated otherwise by notes. The absence of (C) indicates instruction available in native mode only. Opcodes in parentheses execute in 9200/9300 compatibility mode only.

Table A—2. Instruction Repertoire (Part 11 of 16)

Listing By Machine Code		
Machine Code	**Mnemonic**	**Instruction Name**
03	STR	Service timer register – privileged
04	SPM	Set program mask
05	BALR	Branch and link
06	BCTR	Branch on count
07	(C)BCR	Branch on condition (Native and 360/20 Modes)
08	(F)SSK	Set storage key – privileged
09	(F)ISK	Insert storage key – privileged
0A	SVC	Supervisor call
0D	(C)BASR	Branch and store (360/20 Mode only)
10	(F)LPR	Load positive
11	(F)LNR	Load negative
12	LTR	Load and test
13	(F)LCR	Load complement
14	NR	AND
15	CLR	Compare logical
16	OR	OR
17	XR	Exclusive OR
18	LR	Load
19	CR	Compare
1A	(C)AR	Add (Native and 360/20 Modes)
1B	(C)SR	Subtract (Native and 360/20 Modes)
1C	(F)MR	Multiply
1D	(F)DR	Divide
1E	(F)ALR	Add logical
1F	(F)SLR	Subtract logical
20	(F)LPDR	Load positive (long)
21	(F)LNDR	Load negative (long)
22	(F)LTDR	Load and test (long)
23	(F)LCDR	Load complement (long)

Table A—2. Instruction Repertoire (Part 12 of 16)

Listing By Machine Code		
Machine Code	**Mnemonic**	**Instruction Name**
24	(F)HDR	Halve (long)
28	(F)LDR	Load (long)
29	(F)CDR	Compare (long)
2A	(F)ADR	Add normalized (long)
2B	(F)SDR	Subtract normalized (long)
2C	(F)MDR	Multiply (long)
2D	(F)DDR	Divide (long)
2E	(F)AWR	Add unnormalized (long)
2F	(F)SWR	Subtract unnormalized (long)
30	(F)LPER	Load positive (short)
31	(F)LNER	Load negative (short)
32	(F)LTER	Load and test (short)
33	(F)LCER	Load complement (short)
34	(F)HER	Halve (short)
38	(F)LER	Load (short)
39	(F)CER	Compare (short)
3A	(F)AER	Add normalized (short)
3B	(F)SER	Subtract normalized (short)
3C	(F)MER	Multiply (short)
3D	(F)DER	Divide (short)
3E	(F)AUR	Add unnormalized (short)
3F	(F)SUR	Subtract unnormalized (short)
40	(C)STH	Store half word
41	LA	Load address
42	STC	Store character
43	IC	Insert character
44	EX	Execute
45	(C)BAL	Branch and link (Native and 9200/9300 Modes)
46	BCT	Branch on count

Table A—2. Instruction Repertoire (Part 13 of 16)

Listing By Machine Code			
Machine Code	**Mnemonic**	**Instruction Name**	
47	(C)BC	Branch on condition	
48	(C)LH	Load half-word	
49	(C)CH	Compare half-word	
4A	(C)AH	Add half-word	(Native and 360/20 Modes)
4B	(C)SH	Subtract half-word	(Native and 360/20 Modes)
4C	(F)MH	Multiply half-word	
4D	(C)BAS	Branch and store	(360/20 Mode only)
4E	CVD	Convert to decimal	
4F	CVB	Convert to binary	
50	ST	Store	
54	N	AND	
55	CL	Compare logical	
56	O	OR	
57	X	Exclusive OR	
58	L	Load	
59	C	Compare	
5A	A	Add	
5B	S	Subtract	
5C	M	Multiply	
5D	D	Divide	
5E	(F)AL	Add logical	
5F	(F)SL	Subtract logical	
60	(F)STD	Store (long)	
68	(F)LD	Load (long)	
69	(F)CD	Compare (long)	
6A	(F)AD	Add normalized (long)	
6B	(F)SD	Subtract normalized (long)	
6C	(F)MD	Multiply (long)	
6D	(F)DD	Divide (long)	

Table A—2. Instruction Repertoire (Part 14 of 16)

Listing By Machine Code		
Machine Code	**Mnemonic**	**Instruction Name**
6E	(F)AW	Add unnormalized (long)
6F	(F)SW	Subtract unnormalized (long)
70	(F)STE	Store (short)
78	(F)LE	Load (short)
79	(F)CE	Compare (short)
7A	(F)AE	Add normalized (short)
7B	(F)SE	Subtract normalized (short)
7C	(F)ME	Multiply (short)
7D	(F)DE	Divide (short)
7E	(F)AU	Add unnormalized (short)
7F	(F)SU	Subtract unnormalized (short)
80	SSM	Set system mask – privileged
82	LPSW	Load PSW – privileged
83	DIAG	Diagnose – privileged
86	(F)BXH	Branch on index high
87	(F)BXLE	Branch on index low or equal
88	SRL	Shift right single logical
89	SLL	Shift left single logical
8A	(F)SRA	Shift right single
8B	(F)SLA	Shift left single
8C	(F)SRDL	Shift right double logical
8D	(F)SLDL	Shift left double logical
8E	(F)SRDA	Shift right double
8F	(F)SLDA	Shift left double
90	STM	Store multiple
91	(C)TM	Test under mask
92	(C)MVI	Move immediate
93	(F)TS	Test and set
94	(C)NI	AND

Table A—2. Instruction Repertoire (Part 15 of 16)

Listing By Machine Code			
Machine Code	**Mnemonic**	**Instruction Name**	
95	(C)CLI	Compare logical	
96	(C)OI	OR	
97	XI	Exclusive OR	
98	LM	Load multiple	
99	HPR	Halt and proceed – privileged	
9A	AI	Add immediate	
9C	SIO	Start I/O – privileged	
A2	SSFS	SOFTSCOPE forward scan – privileged	
A3	SSRS	SOFTSCOPE reverse scan – privileged	
(A6)	(C)AI	Add immediate	(9200/9300 Mode only)
(AA)	(C)AH	Add half word	(9200/9300 Mode only)
(AB)	(C)SH	Subtract half word	(9200/9300 Mode only)
B0	SSTM	Supervisor store multiple – privileged	
B1	LCS	Load control storage – privileged	
B8	SLM	Supervisor load multiple – privileged	
D1	(C)MVN	Move numerics	
D2	(C)MVC	Move	
D3	(C)MVZ	Move zones	(Native and 360/20 Modes)
D4	(C)NC	AND	(Native and 9200/9300 Modes)
D5	(C)CLC	Compare logical	
D6	(C)OC	OR	(Native and 9200/9300 Modes)
D7	XC	Exclusive OR	
DC	(C)TR	Translate	
DD	TRT	Translate and test	
DE	(C)ED	Edit	
DF	(F)EDMK	Edit and mark	
F1	(C)MVO	Move with off set	
F2	(C)PACK	Pack	
F3	(C)UNPK	Unpack	

Table A—2. Instruction Repertoire (Part 16 of 16)

Listing By Machine Code		
Machine Code	**Mnemonic**	**Instruction Name**
F8	(C)ZAP	Zero and add
F9	(C)CP	Compare decimal
FA	(C)AP	Add decimal
FB	(C)SP	Subtract decimal
FC	(C)MP	Multiply decimal
FD	(C)DP	Divide decimal

NOTES:

1. Tag symbol (F) before mnemonic indicates instructions that are added as features.

2. Tag symbol (C) before mnemonic indicates instruction available in native mode and in 9200/9300 and 360/20 compatibility modes, unless indicated otherwise by notes. The absence of (C) indicates instruction available in native mode only.

3. Opcodes in parentheses execute in 9200/9300 compatibility mode only.

Table A—3. Extended Mnemonic Branch Codes

RR-Type Instructions		RX-Type Instructions		BC Equivalent		Function
Mnemonic Code	Hexadecimal Operation Code m_1	Mnemonic Code	Hexadecimal Operation Code m_1	Explicit Form		
BR	07 F	–	–	BCR	$15,r_2$	Branch unconditionally
NOPR	07 0	–	–	BCR	$0,r_2$	No operation
–	–	B	47 F	BC	$15,d_2(x_2,b_2)$	Branch unconditionally
–	–	NOP	47 0	BC	$0,d_2(x_2,b_2)$	No operation
Used After Comparison Instructions						
BHR	07 2	BH	47 2	BC	$2,d_2(x_2,b_2)$	Branch if high
BLR	07 4	BL	47 4	BC	$4,d_2(x_2,b_2)$	Branch if low
BER	07 8	BE	47 8	BC	$8,d_2(x_2,b_2)$	Branch if equal
BNHR	07 D	BNH	47 D	BC	$13,d_2(x_2,b_2)$	Branch if not high
BNLR	07 B	BNL	47 B	BC	$11,d_2(x_2,b_2)$	Branch if not low
BNER	07 7	BNE	47 7	BC	$7,d_2(x_2,b_2)$	Branch if not equal
Used After Test-Under-Mask Instructions						
BOR	07 1	BO	47 1	BC	$1,d_2(x_2,b_2)$	Branch if all ones
BZR	07 8	BZ	47 8	BC	$8,d_2(x_2,b_2)$	Branch if all zeros
BMR	07 4	BM	47 4	BC	$4,d_2(x_2,b_2)$	Branch if mixed
BNOR	07 E	BNO	47 E	BC	$14,d_2(x_2,b_2)$	Branch if not all ones
BNZR	07 7	BNZ	47 7	BC	$7,d_2(x_2,b_2)$	Branch if not all zeros
BNMR	07 B	BNM	47 B	BC	$11,d_2(x_2,b_2)$	Branch if not mixed
Used After Arithmetic Instructions						
BOR	07 1	BO	47 1	BC	$1,d_2(x_2,b_2)$	Branch if overflow
BZR	07 8	BZ	47 8	BC	$8,d_2(x_2,b_2)$	Branch if zero
BMR	07 4	BM	47 4	BC	$4,d_2(x_2,b_2)$	Branch if minus
BPR	07 2	BP	47 2	BC	$2,d_2(x_2,b_2)$	Branch if positive
BNOR	07 E	BNO	47 E	BC	$14,d_2(x_2,b_2)$	Branch if not overflow
BNZR	07 7	BNZ	47 7	BC	$7,d_2(x_2,b_2)$	Branch if not zero
BNMR	07 B	BNM	47 B	BC	$11,d_2(x_2,b_2)$	Branch if not minus
BNPR	07 D	BNP	47 D	BC	$13,d_2(x_2,b_2)$	Branch if not positive

Table A–4. Summary of Operators

Classification	Operator	Description	Hierarchy
Arithmetic operators	*/	A*/B is equivalent to A*2^B	6
	//	Covered quotient, A//B is equivalent to (A+B–1)/B	5
	/	A/B means arithmetic quotient of A and B.	5
	*	A*B means arithmetic product of A and B.	5
	–	A–B means arithmetic difference of A and B.	4
	+	A+B means arithmetic sum of A and B.	4
Logical operators	**	A**B means logical product AND of A and B.	3
	++	A++B means logical sum OR of A and B.	2
	––	A––B means logical difference XOR of A and B.	2
Relational operators	=	A=B has value 1 if true; has value 0 if false.	1
	>	A>B has value 1 if true; has value 0 if false.	1
	<	A<B has value 1 if true; has value 0 if false.	1

Table A—5. Comparison of Terms

Term	Examples	
SDTs ■ Can be used in the 1st or 2nd operands. ■ May be used in application instructions and in assembler directions.	CLI MVI MVC	AREA10,$\underbrace{\text{10}}_{\text{SDT}}$ AREAB,$\underbrace{\text{X'C2'}}_{\text{SDT}}$ $\underbrace{\text{33}}_{\text{SDT}}\underbrace{\text{(10R5)}}_{\text{SDT}},\underbrace{\text{3}}_{\text{SDT}}\text{(R8)}$
Literals ■ May only be used in the last operand. ■ May not be used in assembler directives. ■ Literals are preceded by an equal (=) sign.	MVC MVC CLC	AREA10,$\underbrace{\text{=C'10'}}_{\text{Literal}}$ AREA10,$\underbrace{\text{=X'F1F0'}}_{\text{Literal}}$ ONSW,$\underbrace{\text{=B'11111111'}}_{\text{Literal}}$
Symbols for constants ■ May be used in the 1st or 2nd operands. ■ May be used in application instructions and in assembler directives.	AREA10 NO10 MOVE10	DS CL2 DC C'10' MVC $\underbrace{\text{AREA10,NO10}}_{\text{symbols}}$

Table A—6. Characteristics of Constant and Storage Definition Type Codes

Type Code	Constant or Storage Type	Alignment	Source Code Specification		Storage Format	Truncation or Padding	Length in Bytes		
							Implied	Minimum Explicit	Maximum Explicit *
C	Character	None	Characters	C' '	Character	Right	Variable	1	256 (DC) 65,535 (DS)
X	Hexadecimal	None	Hexadecimal digits	X' '	Hexadecimal	Left	Variable	1	256 (DC) 65,535 (DS)
B	Binary	None	Binary digits	B' '	Binary	Left	Variable	1	256
P	Packed decima	None	Decimal digits	P' '	Packed decimal	Left	Variable	1	16
Z	Zoned decimal	None	Decimal digits	Z' '	Character	Left	Variable	1	16
H	Half word, fixed point	Half word	Decimal digits	H' '	Fixed-point binary	Left	2	1	8
F	Full word, fixed point	Full word	Decimal digits	F' '	Fixed-point binary	Left	4	1	8
Y	Half-word address	Half word	Expression	Y( )	Binary	Left	2	1	2
A	Full-word address	Full word	Expression	A( )	Binary	Left	4	1	4
S	Base and displacement	Half word	One or two expressions	S( )	Base and displacement	None	2	2	2
V	External address	Full word	Relocatable symbol	V( )	Binary	Left	4	3	4
E	Full word, floating point	Full word	Decimal digits	E' '	Floating-point binary normalized	Right	4	1	8
D	Double word, floating point	Double word	Decimal digits	D' '	Floating-point binary normalized	Right	8	1	8

*The maximum explicit length in bytes is that total length produced by the explicit length factor times the duplication factor.

Table A–7. PROC, MACRO, and Call Instruction Comparison

PROC CONSTRUCTION

	LABEL	ΔOPERATIONΔ	OPERAND
HEADING	[&symbol] call-name	PROC NAME	[&pos,n] [pos-0] [,&key$_1$=,...,&key$_m$=]
BODY	[{symbol, &symbol, .symbol}]	mnemonic-code . . . mnemonic-code	operands . . . operands
TRAILER	unused	END	unused

MACRO CONSTRUCTION

	LABEL	ΔOPERATIONΔ	OPERAND
HEADING	unused	MACRO	unused
	[&symbol]	call-name	[&pos$_1$,...,&pos$_n$][,&key$_1$=,...,&key$_m$=]
BODY	[{symbol, &symbol, .symbol}]	mnemonic-code . . . mnemonic-code	operands . . . operands
TRAILER	unused	MEND	unused

CALL INSTRUCTION FORMAT

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	call-name	[p$_1$,p$_2$,...,p$_{252}$]

Explanation:

- Addressing

  A storage location outside the range of the installed storage is referenced by a program-specified address.

- Data

  — An invalid sign or digit code is detected in decimal operands.

  — Fields in decimal arithmetic overlap incorrectly.

  — The first operand of the *multiply decimal* instruction does not have a sufficient number of high-order zero digits.

- Decimal Divide

  The quotient of a *divide decimal* instruction exceeds the capacity of the quotient part of the first operand field.

- Decimal Overflow

  The result of an *add decimal, subtract decimal, or zero and add* instruction exceeds the capacity of the first operand location.

- Execute

  The subject instruction of an *execute* instruction is an *execute* instruction.

- Exponent Overflow

  The final characteristic resulting from a floating-point arithmetic operand exceeds 127.

- Exponent Underflow

  The final characteristic resulting from a floating-point arithmetic operation is less than zero.

Table A—8. Check-off Table Terms

General			
OPCODE		FORMAT TYPE	OBJECT INST. LGTH. (BYTES)
MNEM.	HEX.		

Condition Codes
☐ IF RESULT = 0, SET TO 0
☐ IF RESULT < 0, SET TO 1
☐ IF RESULT > 0, SET TO 2
☐ IF OVERFLOW, SET TO 3
☐ UNCHANGED

Possible Program Exceptions	
☐ ADDRESSING	☐ PROTECTION
☐ DATA (INVALID SIGN/DIGIT)	☐ SIGNIFICANCE
☐ DECIMAL DIVIDE	☐ SPECIFICATION:
☐ DECIMAL OVERFLOW	☐ NOT A FLOATING-POINT REGISTER
☐ EXECUTE	☐ OP 1 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT OVERFLOW	☐ OP 2 NOT ON HALF-WORD BOUNDARY
☐ EXPONENT UNDERFLOW	☐ OP 2 NOT ON FULL-WORD BOUNDARY
☐ FIXED-POINT DIVIDE	☐ OP 2 NOT ON DOUBLE-WORD BOUNDARY
☐ FIXED-POINT OVERFLOW	☐ OP 1 NOT EVEN NUMBERED REGISTER
☐ FLOATING-POINT DIVIDE	☐ OP 1 NOT ODD NUMBERED REGISTER
☐ OPERATION	☐ NONE

Explanation:

- Fixed-Point Divide

  The quotient of a fixed-point divide operation exceeds the capacity of the first operand (including division by zero), or the result of a *convert to binary* instruction exceeds 31 bits.

- Fixed-Point Overflow

  A fixed-point add or subtract operation exceeds the capacity of the first operand field.

- Floating-Point Divide

  The divisor fraction in a floating-point divide operation is equal to zero.

- Operation

  An illegal operation has been attempted or an operation using a noninstalled processor feature has been attempted.

- Protection

  A storage protection violation occurs on a program-generated address, when the protection feature is installed.

- Significance

  The final fraction resulting from a floating-point addition or subtraction is equal to zero.

- Specification

  — The unit of information referenced is not on an appropriate boundary.

  — An invalid modifier field is specified in the STR instruction.

  — The R_1 field of an instruction which uses an even/odd pair of registers (64-bit operand) does not specify an even register.

  — A floating-point register other than 0, 2, 4, or 6 is specified.

  — A multiplicand or divisor in decimal arithmetic exceeds 15 digits and sign.

  — The first operand field is shorter than, or equal in length to, the second operand in *decimal multiply* and *decimal divide* instructions.

Appendix B. Character Set Code References

Table B—1. Punched Card, ASCII, and EBCDIC Codes (Part 1 of 5)

Character	Printed Symbol	Card Punches	ASCII		EBCDIC	
			Hexadecimal	Decimal	Hexadecimal	Decimal
Letters						
Uppercase A	A	12–1	41	65	C1	193
Uppercase B	B	12–2	42	66	C2	194
Uppercase C	C	12–3	43	67	C3	195
Uppercase D	D	12–4	44	68	C4	196
Uppercase E	E	12–5	45	69	C5	197
Uppercase F	F	12–6	46	70	C6	198
Uppercase G	G	12–7	47	71	C7	199
Uppercase H	H	12–8	48	72	C8	200
Uppercase I	I	12–9	49	73	C9	201
Uppercase J	J	11–1	4A	74	D1	209
Uppercase K	K	11–2	4B	75	D2	210
Uppercase L	L	11–3	4C	76	D3	211
Uppercase M	M	11–4	4D	77	D4	212
Uppercase N	N	11–5	4E	78	D5	213
Uppercase O	O	11–6	4F	79	D6	214
Uppercase P	P	11–7	50	80	D7	215
Uppercase Q	Q	11–8	51	81	D8	216
Uppercase R	R	11–9	52	82	D9	217
Uppercase S	S	0–2	53	83	E2	226
Uppercase T	T	0–3	54	84	E3	227
Uppercase U	U	0–4	55	85	E4	228
Uppercase V	V	0–5	56	86	E5	229
Uppercase W	W	0–6	57	87	E6	230
Uppercase X	X	0–7	58	88	E7	231
Uppercase Y	Y	0–8	59	89	E8	232
Uppercase Z	Z	0–9	5A	90	E9	233
Lowercase a	a	12–0–1	61	97	81	129
Lowercase b	b	12–0–2	62	98	82	130
Lowercase c	c	12–0–3	63	99	83	131

Table B—1. Punched Card, ASCII, and EBCDIC Codes (Part 2 of 5)

Character	Printed Symbol	Card Punches	ASCII		EBCDIC	
			Hexadecimal	Decimal	Hexadecimal	Decimal
Lowercase d	d	12–0–4	64	100	84	132
Lowercase e	e	12–0–5	65	101	85	133
Lowercase f	f	12–0–6	66	102	86	134
Lowercase g	g	12–0–7	67	103	87	135
Lowercase h	h	12–0–8	68	104	88	136
Lowercase i	i	12–0–9	69	105	89	137
Lowercase j	j	12–11–1	6A	106	91	145
Lowercase k	k	12–11–2	6B	107	92	146
Lowercase l	l	12–11–3	6C	108	93	147
Lowercase m	m	12–11–4	6D	109	94	148
Lowercase n	n	12–11–5	6E	110	95	149
Lowercase o	o	12–11–6	6F	111	96	150
Lowercase p	p	12–11–7	70	112	97	151
Lowercase q	q	12–11–8	71	113	98	152
Lowercase r	r	12–11–9	72	114	99	153
Lowercase s	s	11–0–2	73	115	A2	162
Lowercase t	t	11–0–3	74	116	A3	163
Lowercase u	u	11–0–4	75	117	A4	164
Lowercase v	v	11–0–5	76	118	A5	165
Lowercase w	w	11–0–6	77	119	A6	166
Lowercase x	x	11–0–7	78	120	A7	167
Lowercase y	y	11–0–8	79	121	A8	168
Lowercase z	z	11–0–9	7A	122	A9	169
Numerals						
0	0	0	30	48	F0	240
1	1	1	31	49	F1	241
2	2	2	32	50	F2	242
3	3	3	33	51	F3	243
4	4	4	34	52	F4	244
5	5	5	35	53	F5	245
6	6	6	36	54	F6	246

Table B—1. Punched Card, ASCII, and EBCDIC Codes (Part 3 of 5)

Character	Printed Symbol	Card Punches	ASCII		EBCDIC	
			Hexadecimal	Decimal	Hexadecimal	Decimal
7	7	7	37	55	F7	247
8	8	8	38	56	F8	248
9	9	9	39	57	F9	249
Symbols						
Exclamation point	!	12–8–7	21	33	4F	79
Quotation mark, dieresis	"	8–7	22	34	7F	127
Number sign, pound sign	#	8–3	23	35	7B	123
Dollar sign	$	11–8–3	24	36	5B	91
Percent sign	%	0–8–4	25	37	6C	108
Ampersand	&	12	26	38	50	80
Apostrophe, acute accent	'	8–5	27	39	7D	125
Opening parenthesis	(	12–8–5	28	40	4D	77
Closing parenthesis	)	11–8–5	29	41	5D	93
Asterisk	*	11–8–4	2A	42	5C	92
Plus sign	+	12–8–6	2B	43	4E	78
Comma, cedilla	,	0–8–3	2C	44	6B	107
Minus sign, hyphen	–	11	2D	45	60	96
Period, decimal point	.	12–8–3	2E	46	4B	75
Slash, virgule, solidus	/	0–1	2F	47	61	97
Colon	:	8–2	3A	58	7A	122
Semicolon	;	11–8–6	3B	59	5E	94
Less than	<	12–8–4	3C	60	4C	76
Equal sign	=	8–6	3D	61	7E	126
Greater than	>	0–8–6	3E	62	6E	110
Question mark	?	0–8–7	3F	63	6F	111
Commercial at symbol	@	8–4	40	64	7C	124
Opening bracket	[	12–8–2	5B	91	4A	74
Closing bracket	]	11–8–2	5D	93	5A	90
Reverse slash	\	0–8–2	5C	92	E0	224
Circumflex	∧	11–8–7	5E	94	5F	95

Table B—1. Punched Card, ASCII, and EBCDIC Codes (Part 4 of 5)

Character	Printed Symbol	Card Punches	ASCII		EBCDIC	
			Hexadecimal	Decimal	Hexadecimal	Decimal
Underline	—	0–8–5	5F	95	6D	109
Grave accent	`	8–1	60	96	79	121
Opening brace	{	12–0	7B	123	C0	192
Closing brace	}	11–0	7D	125	D0	208
Vertical line	¦	12–11	7C	124	6A	106
Overline, tilde	~	11–0–1	7E	126	A1	161

Character	Card Punches	ASCII		EBCDIC	
		Hexadecimal	Decimal	Hexadecimal	Decimal
Nonprintable Characters					
ACK (Acknowledge)	0–9–8–6	06	6	2E	46
BEL (Bell)	0–9–8–7	07	7	2F	47
BS (Backspace)	11–9–6	08	8	16	22
CAN (Cancel)	11–9–8	18	24	18	24
CR (Carriage return)	12–9–8–5	0D	13	0D	13
DC1 (Device control 1)	11–9–1	11	17	11	17
DC2 (Device control 2)	11–9–2	12	18	12	18
DC3 (Device control 3)	11–9–3	13	19	13	19
DC4 (Device control 4)	9–8–4	14	20	3C	60
DEL (Delete)	12–9–7	7F	127	07	7
DLE (Data link escape)	12–11–9–8–1	10	16	10	16
DS (Digit select)	11–0–9–8–1	80	128	20	32
EM (End of medium)	11–9–8–1	19	25	19	25
ENQ (Enquiry)	0–9–8–5	05	5	2D	45
EOT (End of transmission)	9–7	04	4	37	55
ESC (Escape)	0–9–7	1B	27	27	39
ETB (End of transmission block)	0–9–6	17	23	26	38
ETX (End of text)	12–9–3	03	3	03	3
FF (Form feed)	12–9–8–4	0C	12	0C	12
FS (File separator)	11–9–8–4	1C	28	1C	28

Table B—1. Punched Card, ASCII, and EBCDIC Codes (Part 5 of 5)

Character	Card Punches	ASCII		EBCDIC	
		Hexadecimal	Decimal	Hexadecimal	Decimal
FS (Field separator)	0–9–2	82	130	22	34
GS (Group separator)	11–9–8–5	1D	29	1D	29
HT (Horizontal tabulation)	12–9–5	09	9	05	5
LF (Line feed)	0–9–5	0A	10	25	37
NAK (Negative acknowledge)	9–8–5	15	21	3D	61
NUL (Null)	12–0–9–8–1	00	0	00	0
RS (Record separator)	11–9–8–6	1E	30	1E	30
SI (Shift in)	12–9–8–7	0F	15	0F	15
SO (Shift out)	12–9–8–6	0E	14	0E	14
SOH (Start of heading)	12–9–1	01	1	01	1
SOS (Significance start)	0–9–1	81	129	21	33
SP (Space)		20	32	40	64
STX (Start of text)	12–9–2	02	2	02	2
SUB (Substitute)	9–8–7	1A	26	3F	63
SYN (Synchronous idle)	9–2	16	22	32	50
US (Unit separator)	11–9–8–7	1F	31	1F	31
VT (Vertical tabulation)	12–9–8–3	0B	11	0B	11

Table B—2. 90/30 EBCDIC Code Chart

		Bit Positions 0, 1, 2, 3															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Bit Positions 4, 5, 6, 7	**0000**	NUL	DLE	DS①		SP	&	–						{④	}④	\④	0
	0001	SOH	DC1	SOS①				/		a④	j	~④		A	J		1
	0010	STX	DC2	FS①	SYN					b	k	s		B	K	S	2
	0011	ETX	DC3							c	l	t		C	L	T	3
	0100									d	m	u		D	M	U	4
	0101	HT		LF						e	n	v		E	N	V	5
	0110		BS	ETB						f	o	w		F	O	W	6
	0111	DEL		ESC	EOT					g	p	x		G	P	X	7
	1000		CAN							h	q	y		H	Q	Y	8
	1001		EM						\`④	i	r	z		I	R	Z	9
	1010					[	]	¦③	:								
	1011	VT				.	$	,	#								
	1100	FF	FS		DC4	<	*	%	@								
	1101	CR	GS	ENQ	NAK	(	)	_	'								
	1110	SO	RS	ACK		+	;	>	=								
	1111	SI	US	BEL	SUB	!②	¬②	?	"								

NOTES:

EBCDIC bits are numbered from the left in ascending numerical order: 0 1 2 3 4 5 6 7. Some graphic card code and hexadecimal assignments may differ depending on the device, language, application, and installation policy.

① DS, SOS, FS are the control characters for the EDIT instruction and have been assigned for ASCII mode processing so as not to conflict with the corresponding character positions previously assigned in the EBCDIC chart. As these characters are not outside the range as defined in *American National Standard, X3.4 – 1968,* they must not appear in external storage media, such as ANSI standard tapes. This presents no difficulty due to the nature of the EDIT instruction.

② The following optional graphics can be substituted in the character set:

$\wedge$ for ¬

| for !

③ For 63-character printers, the following substitution is made:

\ for ¦

④ The lowercase alphabet and indicated graphics are introduced by use of the type 0768—02 printer, which prints a 94-character set.

Table B—3. ASCII Character Code Chart

	Bit Positions 7, 6, 5							
Bit Positions 4, 3, 2, 1	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	! ①	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	¦
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	∧ ①	n	~
1111	SI	US	/	?	O	_	o	DEL

② ③

④

NOTES:

ASCII bits are numbered from the left in descending numerical order: 7 6 5 4 3 2 1. Some graphic card code and hexadecimal assignments may differ depending on the device, language, application, and installation policy.

① The following optional graphics can be substituted in the following set:

¬ for ∧

| for !

② Sixty-three printable character set.

③ Graphics available by use of the type 0768-02 printer which prints a 94-character set (DEL is not a graphic)

④ Ninety-four printable character set.

Control Character Mnemonics

ACK – Acknowledge
BEL – Bell
BS – Backspace
CAN – Cancel
CR – Carriage return
DC1 – Device control 1
DC2 – Device control 2
DC3 – Device control 3
DC4 – Device control 4
DEL – Delete
DLE – Data link escape
DS – Digit select
EM – End of medium
ENQ – Enquiry
EOT – End of transmission
ESC – Escape
ETB – End of transmission block
ETX – End of text
FF – Form feed
FS – Field separator
GS – Group separator
HT – Horizontal tab
LF – Line field
NAK – Negative acknowledge
NUL – Null
RS – Record separator
SI – Shift in
SO – Shift out
SOH – Start of heading
SOS – Start of significance
SP – Space
STX – Start of text
SUB – Substitute
SYN – Synchronous idle
US – Unit separator
VT – Vertical tab

Appendix C. Math References

Table C—1. Comparison for Numeric Expressions

Type of Number	Examples	Decimal Values
Character form (unpacked)	F 5 F 0 F 0	500
Zoned decimal (+)	F 5 F 0 C 0	+500
Zoned decimal (—)	F 5 F 0 D 0	—500
Packed decimal (+ only)	5 0 0 F	+500
Packed decimal, signed (+)	5 0 0 C	+500
Packed decimal, signed (—)	5 0 0 D	—500
Hexadecimal (+ only)	0 1 F 4	+500
Floating point (+)	4 3 1 F 4 0 0 0	+500
Floating point (—)	C 3 1 F 4 0 0 0	—500
Binary (+ only)	0000 0001 1111 0100	+500
Binary (+ only)	1111 1110 0000 1100	+65,036
Fixed point (+)	0000 0001 1111 0100	+500
Fixed point (—)	1111 1110 0000 1100	—500

Table C—2. Hexadecimal-Decimal Integer Conversion (Part 1 of 4)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

Table C—2. Hexadecimal-Decimal Integer Conversion (Part 2 of 4)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

Table C—2. Hexadecimal-Decimal Integer Conversion (Part 3 of 4)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

Table C—2. Hexadecimal-Decimal Integer Conversion (Part 4 of 4)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Table C—3. Hexadecimal-Decimal Fraction Conversion

First Digit		Second Digit			Third Digit				Fourth Digit			
Hex.	Decimal	Hex.	Decimal		Hex.	Decimal			Hex.	Decimal		
.0	.0000	.00	.0000	0000	.000	.0000	0000	0000	.0000	.0000	0000	0000
.1	.0625	.01	.0039	0625	.001	.0002	4414	0625	.0001	.0000	1525	8789
.2	.1250	.02	.0078	1250	.002	.0004	8828	1250	.0002	.0000	3051	7578
.3	.1875	.03	.0117	1875	.003	.0007	3242	1875	.0003	.0000	4577	6367
.4	.2500	.04	.0156	2500	.004	.0009	7656	2500	.0004	.0000	6103	5156
.5	.3125	.05	.0195	3125	.005	.0012	2070	3125	.0005	.0000	7629	3945
.6	.3750	.06	.0234	3750	.006	.0014	6486	3750	.0006	.0000	9155	2734
.7	.4375	.07	.0273	4375	.007	.0017	0898	4375	.0007	.0001	0681	1523
.8	.5000	.08	.0312	5000	.008	.0019	5312	5000	.0008	.0001	2207	0313
.9	.5625	.09	.0351	5625	.009	.0021	9726	5625	.0009	.0001	3732	9102
.A	.6250	.0A	.0390	6250	.00A	.0024	4140	6250	.000A	.0001	5258	7891
.B	.6875	.0B	.0429	6875	.00B	.0026	8554	6875	.000B	.0001	6784	6680
.C	.7500	.0C	.0468	7500	.00C	.0029	2968	7500	.000C	.0001	8310	5469
.D	.8125	.0D	.0507	8125	.00D	.0031	7382	8125	.000D	.0001	9836	4258
.E	.8750	.0E	.0546	8750	.00E	.0034	1796	8750	.000E	.0002	1362	3047
.F	.9375	.0F	.0585	9375	.00F	.0036	6210	9375	.000F	.0002	2888	1836

To convert a 4-digit (2-byte) hexadecimal fraction to a decimal fraction, add the values shown in the above table for each of the hexadecimal digits to be converted as illustrated below. The hexadecimal fraction .B5A1 equals the approximate decimal fraction .70948791 from the above table.

.B	from the table equals	.6875
.05	from the table equals	.01953125
.00A	from the table equals	.002441406250
.0001	from the table equals	.000015258789
.B5A1	equals the sum	.709487915039

NOTE:

All values listed are approximate values.

Table C—4. Hexadecimal Addition and Subtraction Table

	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	
01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	01
02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	02
03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	03
04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	04
05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	05
06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	06
07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	07
08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	08
09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	09
0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	0A
0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	0B
0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	0C
0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	0D
0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	0E
0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	10
11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	11
12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	12
13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	13
14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	14
15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	15
16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	16
17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	17
18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	18
19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	19
1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	1A
1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	1B
1C	1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	1C
1D	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	1D
1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	1E
1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	1F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	

Table C—5. Powers of 16

16^n							n
						1	0
						16	1
						256	2
					4	096	3
					65	536	4
				1	048	576	5
				16	777	216	6
				268	435	456	7
			4	294	967	296	8
			68	719	476	736	9
		1	099	511	627	776	10
		17	592	186	044	416	11
		281	474	976	710	656	12
	4	503	599	627	370	496	13
	72	057	594	037	927	936	14
1	152	921	504	606	846	976	15

These powers of 16 are especially useful in determining the value of floating-point numbers.

Table C—6. Powers of 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 45
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

FLOATING-POINT MATH

The floating-point instruction set is added to the instruction repertoire as part of the floating-point control feature. An operation exception results if a floating-point instruction is issued to a processor in which the floating-point control feature has not been installed.

The floating-point instruction set provides for loading, adding, subtracting, comparing, multiplying, dividing, storing, and sign control of short or long format floating-point operands. Four double-word floating-point registers are provided to accommodate storing and loading of results and operands. These registers are numbered 0, 2, 4, and 6. The specification of any other register number results in a specification exception. For long format operands, the entire double-word register is involved in the operation. For short format operands, excluding the product in the *short format multiply* (ME) instruction, only the most significant word of the double-word register is involved in the operation. The least significant word remains unchanged. Separate instructions are provided for operations with long and short format operands.

Each operand is treated as a floating-point number consisting of a biased exponent (characteristic) and a signed fraction (mantissa). The biased exponent is expressed in excess-64 binary notation; the fraction is expressed as a hexadecimal number having an arithmetic point to the left of the high-order digit. The quantity expressed by the full floating-point number is the product of the fraction and the number 16 raised to the power of the biased exponent minus 64 (fraction times 16^{n-64}).

A quantity may be represented with the greatest precision by a floating-point number of a given fraction length when the number is in a "normalized" form. A normalized floating-point number has a nonzero, high-order hexadecimal fraction digit.

An exponent overflow exception develops if, in the result of a floating-point instruction, the characteristic of the result exceeds 127 and the fraction of the result is not zero. An exponent underflow exception develops if the characteristic is less than zero and the fraction of the result is not zero. An exponent overflow exception causes a program interruption. An exponent underflow exception causes a program interruption if the exponent underflow mask bit f the current PSW is 1.

A floating-point number having a zero characteristic, a zero fraction, and a positive (zero) sign is said to be a "true zero" number.

The floating-point instructions are available in RR and RX formats. Therefore, at least one of the operands is contained in one of the floating-point registers. The other operand is located in the same or another register or in main storage. Each main storage address may be specified as relative or absolute.

To increase the precision of certain computations, an additional least significant digit, the guard digit, is carried within the hardware in the intermediate result of the following operations: add-normalized, subtract-normalized, add-unnormalized, subtract-unnormalized, compare, halve, and multiply. In the execution of add-normalized, subtract-normalized, add-unnormalized, subtract-unnormalized, and compare instructions, when a right shift of the fraction is required to equalize two exponents, the last hexadecimal digit to be shifted out of the least significant digit position of the fraction is saved by the processor hardware as the guard digit. The shifted fraction, including the guard digit, is used in computing the intermediate result. In the halve instruction, the least significant bit position of the fraction is saved as the fifteenth digit of the fraction of the intermediate product. If the intermediate result is subsequently normalized, the guard digit is shifted left to become part of the normalized fraction.

SHORT FORM FLOATING-POINT NUMBER

sign	characteristic (exponent)	mantissa (fraction)
	1 7	8 31

LONG FORM FLOATING-POINT NUMBER

sign	characteristic (exponent)	mantissa (fraction)
	1 7	8 63

Floating-Point Addition

Floating-point addition consists of exponent equalization and fraction addition. If the exponents are equal, the fractions are added to form an intermediate sum. If the exponents are unequal, the smaller exponent is subtracted from the larger. The difference indicates the number of hexadecimal digit shifts to the right to be performed on the fraction having the smaller exponent. Each hexadecimal digit shift to the right causes the exponent to be increased by 1. After equalization, the fractions are added to form an intermediate sum.

A carry-over digit of the most significant hexadecimal digit position of the intermediate sum causes the intermediate sum to be shifted right one digit position and the exponent to be increased by 1. If an exponent overflow condition occurs, the resultant floating-point number consists of a normalized and correct fraction, a correct sign, and an exponent which is 128 less than the correct value.

- Normalization

  The intermediate sum is composed of 14 hexadecimal digits, a guard digit, and a possible carry-over digit. If any most significant digits of the intermediate sum are zero, the fraction including the guard digit is shifted left to form a normalized fraction. Vacated least significant digit positions are zero filled, and the exponent is reduced by the number of shifts. If normalization is unnecessary, the guard digit is 1.

- Exponent Underflow

  If normalization causes the exponent to become less than zero, an exponent underflow condition results. If the exponent underflow mask bit (38) of the current program status word (PSW) is 1, the resultant floating-point number has a correct and normalized fraction, a correct sign, and an exponent which is 128 more than the current value. If the exponent underflow mask of the current PSW is zero, the result is a true zero.

- Zero Result

  If the intermediate sum, including the guard digit, is zero, a significance exception exits. If the significance mask bit (39) of the current PSW is 1, the result is not normalized and the exponent remains unchanged. If the significance mask bit of the current PSW is zero and the intermediate sum is zero, the result is made a true zero. Exponent underflow cannot occur for a zero fraction.

- Sign

  The sign of an arithmetic result is determined algebraically. The sign of a result with a zero fraction is always positive.

Floating-Point Division

Floating-point division consists of exponent subtraction and fraction division. The intermediate quotient exponent is obtained by subtracting the exponents of the two operands and increasing the difference by 64.

Both operands are normalized before division. Consequently, the intermediate quotient is correctly normalized or a right shift of one digit position may be required. The exponent of the intermediate result is increased by 1 if the shift is necessary. All operand 1 (r_1) fraction digits are used in forming the quotient, even if the normalized operand 1 fraction is larger than the normalized operand 2 fraction.

If the final quotient exponent excceds 127, an exponent overflow exception results. The quotient consists of the correct and normalized fraction, a correct sign, and an exponent which is 128 less than the correct value.

If the final quotient exponent is less than zero, an exponent underflow condition exists. If the exponent underflow mask bit of the current PSW is 1, the quotient has a correct and normalized fraction, a correct sign, and an exponent which is 128 greater than the correct value. If the exponent underflow mask bit of the current PSW is zero, the result is made a true zero. Underflow does not apply to the intermediate result or the operands during normalization. An exponent underflow exception causes a program interrupt if the exponent underflow mask bit of the current PSW is 1.

Attempted division by a divisor with a zero fraction leaves the dividend unchanged, and a program exception for floating-point divide occurs. When division of a zero dividend is attempted, the quotient fraction is zero. The quotient sign and exponent are made zero and give a true zero result. No program exceptions occur.

Floating-Point Multiplication

Floating-point multiplication consists of exponent addition and fraction multiplication. The exponent of the intermediate product is obtained by adding the exponents of the two operands and reducing the sum by 64.

Both operands are normalized before multiplication and the intermediate product is normalized after multiplication. The intermediate product fraction is truncated to 14 digits and a guard digit before normalization.

If the exponent of the final product exceeds 127, an exponent overflow condition exists. The resultant floating-point number consists of a correct and normalized fraction, a correct sign, and an exponent which is 128 less than the correct value. The overflow condition does not occur for an intermediate product exponent exceeding 127 if the final exponent is brought within range during normalization.

If the final product exponent is less than zero, an exponent underflow condition exists. If the exponent underflow mask bit (38) of the current PSW is 1, the resultant floating-point number has a correct and normalized fraction, a correct sign, and an exponent which is 128 greater than the correct value. If the exponent underflow mask bit of the current PSW is zero, the result is made a true zero. When an underflow characteristic becomes less than zero during normalization before multiplication, an underflow exception is not recognized.

When all digits of the intermediate product are zero, the result is made a true zero.

When the resulting fraction is zero, a program exception for exponent underflow or overflow does not occur.

Appendix D. Source Corrections

GENERAL

The OS/3 assembler supports a source module correction routine. This routine is the same as the one used in the librarian. The correction deck is interchangeable between the assembler and the librarian except the librarian also uses the added COR control statement. The corrections made to the source module are temporary. The corrections are specified by the presence of both the source module input (//△PARAM△IN=module name or the IN=(vol-ser-no, label) for the jproc call), and the correction records in the job control stream. These records must be within the data delimiters (/$ and /*). If there are no records between the data delimiters, no source correction is performed.

There are three control statements associated with the correction routine: sequence (SEQ), recycle (REC), and skip (SKI). To make the source module corrections, the actual source record to be inserted is used as the correction card with the same sequence number as the record to be replaced. Insertions are performed by using at least one correction card (always the first card) with a sequence number falling between the sequence numbers of the records between which the insertion is to be made. Any number of unsequenced correction cards may then follow the first sequence card. Deletions are performed by bypassing one or more original source module records in the old data set, thus eliminating them from being written on the new data set. The SKI and REC statements are used for this function.

PARAM

The PARAM statement specifies the assembler processing options in effect at assembly time and alters the standard default options. If you don't specify assembler options in the control stream of your job, the assembler functions as follows:

- The assembler searches only the system source library file (YSRC) for any source module or copy code referenced.
- It also searches only the system macro library file (YMAC) for any macro references.
- It stores the object module produced in the job run library file (YRUN).
- It prints the source code, object code, cross-references, and diagnostic listings.
- The value of &SYSPARM is equal to a null string.
- Columns 1 and 2 must contain slashes, followed by at least one blank column, and then PARAM followed by at least one blank column. Multiple options are supported for each option separated by commas. The end of the selected options is indicated by a blank column following the last option. All options selected are printed preceding the assembly listing.

Format:

1	10
// △ PARAM △	[COPY= {filename1 | (N) | **YSRC**} [/{filename2 | (N) | **YSRC**}]]
	[,IN=modulename [/{filename | **YSRC**}]]
	[,LIN= {filename1 | (N) | **YMAC**} [/{filename2 | (N) | **YMAC**}]]
	[,LST= {s | ([s1] [,s2] [,s3] [,s4])}]
	[,OUT= {filename | (N) | **YRUN**}]
	[,R0= {YES | **NO**}]
	[,SYSPARM= {'string' | **null-string**}]

PARAM

The parameter definitions are as follows:

COPY=

Enables up to two files to identified as source code module libraries or specifies that no files are to be searched for source code modules. If this option is omitted, YSRC is assumed and is the only file searched for source code module references. Only source code modules can be copied; the source code must be in the standard format and may not contain any COPY, ICTL, MACRO, PROC, or MEND directives.

filename1

Specifies that the file identified as filename1 is searched first for source code modules referenced and, if not found there, then YSRC is searched: filename is any name you specify or the system source library. If filename1 = filename2, then COPY = filename1 will generate the same files to be searched as COPY = /filename2 except that in the first case the order that the files are searched in will be filename1 and then YSRC; whereas in the 2nd case the order will be YSRC and then filename2.

filename1/filename2

Specifies that the file identified as filename1 is searched first. Then, the file identified as filename2 is searched for source code modules referenced. When two filenames are specified for this parameter, the YSRC file is not searched.

filename1/(N)

Specifies only the file identified as filename1 as searched for source code modules referenced, as stated above, if filename1 = filename2, then COPY=filename1/(N) is the same as COPY = (N)/filename2 with only one file searched in either case.

(N)

Specifies no files, not even YSRC, are searched for source code modules referenced. COPY = (N)/(N) is the same as COPY=(N).

IN=

Identifies the name of the source module that is to be assembled and the file in which it resides. If this option is omitted, the source code must be in the control stream.

modulename

Specifies the name of the source module and directs the assembler to search the YSRC file for the module; modulename is the name of the source module and is up to eight characters.

modulename/filename

Specifies the name of the source module and the file in which it resides; filename is any name you supply or the system source library.

LIN=

Enables up to two files to be identified as macro source files or no files to be searched for macro references. If this option is omitted, YMAC is assumed and is the only file searched.

filename1

Identifies the file that is searched for macro references and, if not found there, then YMAC is searched; filename is any name or the name of the system macro library.

PARAM

filename1/filename2

Identifies the two files that are searched for macro references. The file identified as filename1 is searched first, followed by the file identified as filename 2. The YMAC file is not searched.

filename1/(N)

Specifies only the file identified as filename1 is searched for macro references.

(N)

Specifies no files, not even YMAC, are searched for macro references.

LST=

Indicates the type of listing desired. If this option is omitted, source, object, corss-reference, and diagnostic listings are printed.

s

A single specification requiring no parentheses.

([s_1]...[,s_4])

Any s in the series is one of the following:

NC

Specifies that cross-reference listings are suppressed.

ND

Specifies that diagnostic listings are suppressed.

NR

Specifies that the cross-reference listing is to contain only those symbols that have at least one reference each. If specified with the NC option, NC overrides NR.

N Specifies a proc or macro debug mode feature within the OS/3 assembler. When the feature is selected, the output listing shows the following:

DBG

Specifies a proc or macro debug mode feature within the OS/3 assembler. When the feature is selected, the output listing shows the following:

- Results of the expansion of any proc or macro called within the user program, including any conditional assembly directives processed as the result of the expansion itself. Source coding (constants, directives, and instructions) is listed twice and shows any appropriate substitutions. Any statements causing error diagnostics show the exit line in error.

- A proc or macro which produces error diagnostics at the time it is encoded is listed following the END directive; e.g., system errors. A proc or macro is encoded once, but may be called multiple times.

- If an error is detected at both expansion and encoding time, it appears two or more times. Errors detected only at encoding time appear once following the END directive.

- All lines flagged (regardless of their order or appearance) are shown in the diagnostic summary list. Lines flagged at encoding time may or may not be flagged at expansion time.

When this feature is not selected, any errors detected during proc or macro expansion may not show the exact line in error, but rather the vicinity of the item which is flagged.

OUT=

Enables you to specify the file that is to be used to store the object module output by the assembler. If this option is omitted, the object module is generated and stored in YRUN, the system-run library.

filename

Identifies the file that is used as the output file by the assembler; filename is any name or the job run library.

(N)

Specifies that no output file is used by the assembler and, thus, no object module is generated.

RØ=

Permits you to optionally flag all absolute/base displacement fields of instructions that yield values less than 4096 (1000_{16}). Each statement is flagged with an 'ADDRESSABILITY' error flag.

SYSPARM=

Specifies the equivalent of a global SETC symbol, with the value specified in this option. If this option is omitted, the value of &SYSPARM is a null string.

'string'

Specifies a string of one to eight characters enclosed in apostrophes. An apostrophe within the string is represented by two apostrophes but only counts as one in determining the length of the string.

Operational Consideration:

The value established by SYSPARM is available within the assembly, both outside of and within macro definitions. This parameter is referenced as &SYSPARM within assembly statements. Any error in this specification directs the assembler to ignore the specification, and an appropriate error message is printed on the output printer.

SEQ

Function:

Specifies the starting position and the length of the sequence field. If the sequence field is omitted, column 73 is assumed to be the first column of the sequence field and continue to the maximum of eight characters.

Format:

LABEL	ΔOPERATIONΔ	OPERAND	73 SEQUENCE
unused	SEQ	$,, \left\{ \begin{matrix} \text{column position} \\ 73 \end{matrix} \right\}, \left\{ \begin{matrix} \text{content} \\ 00000000 \end{matrix} \right\}$	

Parameters:

column position

Specifies the first column position in the source record where the sequence field begins.

If omitted, column 73 is assumed to be the first column of the sequence field.

content

One- to eight-character value. The length of this value determines the length of the sequence field.

NOTES:

1. *Card column 1 must be blank if the sequence field does not start in card column 1.*

2. *The SEQ card always is the first card in the correction routine.*

REC

Function:

Causes the record pointer for the input module to be repositioned back to the first record in the module. In conjunction with the SKI control statement, it allows rearranging of major segments of the input module. When a REC control statement is processed, records are read from the input module up to and including the record whose sequence number matches the sequence number in the REC control statement field. Then, the record pointer for the input module is reset to the first record in the module. If the sequence field of the REC control statement is blank, repositioning of the record pointer takes place immediately.

Format:

LABEL	ΔOPERATIONΔ	OPERAND	73 SEQUENCE
ignored	REC	unused	[last-sequence-no.]

Parameters:

last-sequence no.

One to eight alphanumeric characters identifying the sequence number of the last input record to be read from the input module.

If omitted, the repositioning function takes place immediately.

NOTES:

1. *Records are replaced one at a time by writing a source statement with a sequence number matching the sequence number of the record to be replaced.*

2. *Records are inserted by writing source correction statements with sequence numbers that fall between the sequence numbers of the input records between which insertion is to take place. Blank sequence fields cause an insertion to take place immediately.*

SKI

Function:

Allows one or more original input module records to be bypassed. Records are read from the input module until a sequence number is detected that matches the sequence number of the SKI command. The skip operation is started and continues until a sequence number that matches the operand field of the SKI command is detected. If the sequence field of the skip command is blank, the function is started immediately.

Format:

LABEL	△ OPERATION △	OPERAND	73 SEQUENCE
ignored	**SKI**	**last-sequence-no.**	**[starting-sequence-no.]**

Parameters:

last-sequence-no.
One to eight alphanumeric characters identifying the sequence number of the last input module record to be bypassed.

starting-sequence-no.
One to eight alphanumeric characters identifying the sequence number of the first source module record to be bypassed.

If omitted, the skip operation is started immediately, starting with the input module record that immediately follows the last record operated on.

Appendix E. System Variable Symbols

System variable symbols automatically generate values or character strings at assembly time. There are seven system variable symbols: &SYSECT, &SYSLIST, &SYSNDX, &SYSDATE, &SYSTIME, &SYSJDATE, and &SYSPARM. The following paragraphs contain the functions of each system variable symbol.

&SYSECT is a system variable symbol used to represent the name of the control section containing a macro instruction.

&SYSECT is assigned a value for each inner and outer macro instruction processed by the assembler. This value is the name of the control section containing the macro instruction. If &SYSECT is referenced in a macro definition, its substituted value is the name of the last CSECT, DSECT, or START directive that occurred prior to the macro instruction. If a named CSECT, DSECT, or START directive did not appear prior to the macro instruction, &SYSECT is assigned a null character value during the processing of the macro definition called by the macro call instruction.

Any CSECT or DSECT directives processed within a macro definition affect the value of &SYSECT for any subsequent inner macro instructions in the definition and for any outer and inner macro instructions that occur outside the current nest of macro definitions. However, the value of &SYSECT remains constant during the processing of a given macro instruction, and it is not affected by CSECT or DSECT directives or inner macro instructions occurring in that macro definition.

&SYSLIST is a system variable symbol.

Within a macro definition in macro format, each positional parameter may be referenced by a name; however, each positional parameter need not be named in the macro prototype statement and may be referenced in terms of its position within the macro instruction operand field by writing the system variable symbol &SYSLIST followed by an expression in parentheses. The value of the expression identifies the position of the parameter in the operand field. The expression may be a SETA symbol or a self-defining term. Therefore, if a macro definition prototype statement has the operand field:

```
&A,&B,&C
```

the first positional parameter is referenced either as &A or &SYSLIST(1), the second is referenced either as &B or &SYSLIST(2), and the third positional parameter is either &C or &SYSLIST(3), and so on. This capability, which is used to index through the positional parameters, treats each parameter in the same way.

A null character string is generated in place of &SYSLIST(m) if m is zero or greater than the number of positional parameters supplied in the macro instruction.

The system variable &SYSLIST may not be used in a mixed-mode (positional and keyword parameters included) macro definition.

&SYSNDX is a system variable symbol.

The assembler maintains a counter that is incremented by 1 each time the assembler encounters a macro instruction. The value of this counter within the first macro is 1. The current value of this counter is supplied as the 4-digit character value of the system variable symbol &SYSNDX each time a macro instruction is encountered. A macro definition that defines labels within the code it generates and that may be called more than once in a single assembly generally creates duplicate definitions of the same label. To avoid this problem, the system variable symbol &SYSNDX may be used as a suffix on the labels defined by the macro definition, so that each time the macro definition is called, it will define a different set of labels.

&SYSDATE is a system variable symbol, which you can reference in your program text or within a macro definition to generate the date your program is assembled. The date is produced in your assembly listing as a character string representing the month, day, and year (mm/dd/yy) the program was assembled. If you:

1. assemble your program;

2. store it in a library; and

3. retrieve the assembled program for execution at a later date —

any &SYSDATE reference in your program references the original assembly date, not the current date when your program is executed.

You specify &SYSDATE as either an operand in a source code statement, which defines a constant (DC), or an operand field literal.

Example:

LABEL (1)	ΔOPERATIONΔ (10)	OPERAND (16) Δ
	.	
	.	
	.	
ASMDATE	DC	C'&SYSDATE'

When this line of source code is assembled, the object code contains the current date.

You can also use the &SYSDATE system variable symbol as a literal.

Example:

	.	
	.	
	.	
	MVC	BUF,C'&SYSDATE'

When this line of source code is executed, the assembly date is moved into a main storage area called BUF.

&SYSTIME is a system variable symbol, which you can reference either in your program text or within a macro definition, to generate the time of day your program is assembled. The date is produced in your assembly listing as a character string representing the hour, minute, and second (hh.mm.ss) the assembly was run. If you:

1. assemble your program;

2. store it in a library; and

3. retrieve the assembled program for execution at another time —

any &SYSTIME reference in your program references the original assembly time, not the current time of execution.

You specify &SYSTIME as either an operand in a source code statement, which defines a constant (DC), or an operand field literal.

Example:

```
         .
         .
         .
ASMTIME  DC    C'&SYSTIME'
```

When this line of source code is assembled, the object code contains the current time.

You can also use the &SYSTIME system variable symbol as a literal.

Example:

```
         .
         .
         .
         MVC   BUF,=C'&SYSTIME'
```

When this line of source code is executed, the assembly time is moved into a main storage area called BUF.

&SYSJDATE is a system variable symbol, which you can reference either in your program text or within a macro definition, to generate the Julian date when your program is assembled. The date is produced in your assembly listing as a character string representing the month, day, year, and Julian value — day of the year (mmddyjjj) the assembly was run. If you:

1. assemble your program;

2. store it in a library; and

3. retrieve the assembled program for execution at another time —

any &SYSJDATE reference in your program references the Julian date of the original assembly.

You specify &SYSJDATE as either an operand in a source code statement, which defines a constant (DC), or an operand field literal.

Example:

```
        .
        .
        .
JULDATE  DC     C'&SYSJDATE'
```

When this line of source code is assembled, the object code contains the Julian date.

You can also use the &SYSJDATE system variable symbol as a literal.

Example:

```
        .
        .
        .
         MVC    BUF,=C'&SYSJDATE'
```

When this line of source code is executed, the Julian date is moved into a main storage area called BUF.

&SYSPARM is a system variable symbol, which you can reference either in your program text or within a macro definition, to generate an 8-byte null character string at assembly time. The string is initially null but can be varied by using the PARAM statement (Section 3) as follows:

LABEL	△OPERATION△	OPERAND
//△PARAM△	SYSPARM='string'	

By using the PARAM statement, you can specify a string of up to eight characters, enclosed in apostrophes. Once you've altered the value of &SYSPARM, any references to &SYSPARM produces the character string you specified in the PARAM statement, not a null character string.

To reference the &SYSPARM system variable symbol, you specify &SYSPARM as either an operand in a source code statement, which defines a constant (DC), or an operand field literal.

Example:

```
   LABEL      △OPERATION△          OPERAND                    △
1              10         16
        .
        .
        .
NULSTRNG DC     C'&SYSPARM'
        .
```

When this line of source code is assembled, the object code contains an 8-byte null character string.

You can also use the &SYSPARM system variable symbol as a literal.

Example:

```
         .
         .
         .
         MVC   BUF,=C'&SYSPARM'
```

If you don't precede this source code statement with a PARAM statement when this line of source code is executed, an 8-byte null character string is moved into a main storage area called BUF.

Appendix F. Attribute References

The assembler assigns certain attributes to symbols and macro call operands that you may refer to in conditional assembly statements. These attributes are: type (T), length (L), scale (S), integer (I), count (K), and number (N).

You can specify attributes in conditional assembly statements to control logic, which in turn can control the sequence and contents of the inline expansion code generated from model statements. Each kind of attribute has a specific purpose, which determines when you use it.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	conditional assembly operation code	$\left\{\begin{matrix}T\\L\\S\\I\\K\\N\end{matrix}\right\}$ ' $\left\{\begin{matrix}\text{symbol}\\ \text{\&symbol}\end{matrix}\right\}$

The attribute notation (T, L, S, I, K, or N) denotes which attribute of a symbol or parameter you are using. The symbol or parameter is a reference to the data or field which possesses the attribute. The operation code must be a conditional assembly operation code except when you are using the length attribute.

The origin of an attribute value is always either a symbol or parameter. Table F—1 gives the restrictions for using a symbol or parameter as the reference to obtain a particular data attribute. Whether a symbol or parameter can be used in an attribute reference depends on where the reference is made. If an attribute reference is made in macro source code (from inside a macro definition), a symbol may be referenced for any data attribute except K or N. A symbol cannot be used in a count or number attribute reference in macro source code because when K or N is used inside a macro definition the only data that can be referenced is an operand field in the macro instruction call. To reference an operand field to obtain the K or N attribute you can use a symbolic parameter or &SYSLIST; this also applies to the T, L, S, and I attributes. A SET symbol and the system variable symbols listed in Table F—1 can only be used in the T and K attribute references when in macro source code. You can get all but K or N attributes of a symbol in program source code along with all of the other attributes by using the symbol in the attribute reference. Macro instruction operands cannot be referenced from program source code so a symbolic parameter or &SYSLIST cannot be part of an attribute reference in program source code. However, a SET symbol and the system variable listed in Table F—1 can be used in an attribute reference in program source code.

Table F–1. Valid Attribute Reference Applications

ATTRIBUTE						REFERENCE	LOCATION
T	L	S	I	K	N		
√	√	√	√			Symbol	Macro Source Code
√				√		Set Symbol	
√	√	√	√	√	√	Symbolic Parameter	
√	√	√	√	√	√	&SYSLIST	
√				√		&SYSNDX, &SYSPARM, &SYSJDATE, &SYSECT, and &SYSTIME	
√	√	√	√			Symbol	Program Source Code
√				√		SET Symbol	
√				√		&SYSPARM, &SYSDATE, &SYSJDATE, and &SYSTIME	

√ = Valid Application

There are two requirements that must be met before using symbols in attribute references. First, the symbol must appear either in the operand field of an EXTRN directive used outside of a macro, or in the label field of at lease one assembler directive or instruction outside a macro. Second, there must not be any variable symbol in the source line in whose label field the symbol appears. In regards to the call operand attributes, you must abide by the following criteria: the same as previously mentioned, with the addition that the operand must be a symbol and it may not be one generated by variable symbol replacement. The attributes of the operand are really the attributes of the symbol itself. A nested call operand may be a symbolic parameter whose attributes are then the same as the corresponding outer operand. You can not use a length attribute if the type attribute is J, M, N, O T, or U.

Since a call operand may be a sublist, you can also refer to attributes of a sublist or each individual parameter in the sublist. When you refer to these attributes, they will be assigned the same value as the first parameter in the sublist.

You can refer to attributes on conditional directives both inside and outside of macros. Symbols that appear in the label field of instructions generated by a macro are not assigned attributes.

Type attributes

You can use the type attribute to test for the characteristic of the operand or symbol. This is done by writing a T' followed by the symbol or symbolic parameter to be tested. This can also be used in SETC directive operand fields or as character expressions in SETB and AIF directive operand fields. Table F—2 summarizes the type attributes and the circumstances under which they are produced.

Table F–2. Type Attributes of Symbols (Part 1 of 2)

Type	Symbol Definition	Length Specification	Alignment
A	Type A address constant	Implied	Full-word
B	Binary constant	Implied or explicit	Not applicable
C	Character constant	Implied or explicit	Not applicable
D	Double-word floating-point constant	Implied	Double-word
E	Full-word floating-point constant	Implied	Full-word
F	Full-word fixed-point constant	Implied	Full-word
G	Fixed-point constant	Explicit	Not applicable
H	Half-word fixed-point constant	Implied	Half-word
I	Machine instruction	Implied	Half-word
J	Control section name	Not applicable	Double-word
K	Floating-point constant	Explicit	Not applicable
M	Macro instruction	Not applicable	Not applicable
N (1)	Self-defining term	Not applicable	Not applicable
O (1)	Omitted operand	Not applicable	Not applicable
P	Packed decimal constant	Implied or explicit	Not applicable
R	Unaligned address constant (A, S, V, or Y)	Explicit	Not applicable
S	Type S address constant	Implied	Half-word
T	External symbol	Not applicable	Not applicable
U (2)	Type not available	Not applicable	Not applicable

Table F–2. Type Attributes of Symbols (Part 2 of 2)

Type	Symbol Definition	Length Specification	Alignment
V	Type V address constant	Implied	Full-word
W	CCW statement	Implied	Double-word
X	Hexadecimal constant	Explicit or implied	Not applicable
Y	Type Y address constant	Implied	Half-word
Z	Zoned decimal constant	Explicit or implied	Not applicable

NOTES:

1. This type attribute is produced only for macro instruction operands.
2. Type cannot be assigned. It is produced for inner and outer macro instruction operands that cannot be assigned any other attribute, as well as for literals appearing as macro instruction operands, symbols appearing in the label field of LTORG, ORG, or EQU directives, symbols appearing more than once in a source statement label field, and symbols appearing in the label field of DC or DS directives containing expressions or variable symbols in the modifier subfields. The latter is true even if the modifier subfield expression consists solely of self-defining terms.

Length Attributes

You can reference the length attribute by writing an L' followed by the symbol or parameter whose attribute you want. The length attribute has a numeric value, which refers to the number of bytes assigned by the assembler to a data field. If the length attribute value is required for conditional (preassembly) processing, the symbol you specify in the attribute reference must appear in the label field of a statement in open source code. The operand field of that statement must contain a self-defining term.

The length modifier or length field must not be coded as a multiterm expression because the assembler does not evaluate this expression until assembly time.

When the length attribute is used in conditional assembly statements, it can be specified only within an expression. Examples: L'&P(4), L'&VARY(1,2), L'&SYSLIST(5).

When a length attribute reference is specified in open source code, it is not available for use in conditional assembly statements.

An L' cannot be generated directly by a macro or proc. It can be done indirectly as follows:

LABEL (1)	ΔOPERATIONΔ (10)	OPERAND (16) Δ
	LCLC	&A,&B
&A	SETC	'Z'
&B	SETC	'L'
	MVC	&A.(&B&A),X

After generation this would result in:

MVC Z(L'Z),X

Scale Attributes

You can reference scale attributes of variable symbols by coding an S' followed by the desired symbol. Scaling attributes are available only for labels of statements defining fixed-point or floating-point constants. This restricts them to H, F, D, E, P, type Z, type K, and type G constants in the OS/3 assembler. The scaling attribute is the value you have assigned for the scale modifier of a fixed or floating-point constant. This modifier is an integer used to assign a number of bits in an unnormalized constant for the fractional portion of the constant. For example, the scale modifier of a DC statement such as HF8'—19.788' would be 8, since it is specifying 8 bits for the fractional part of the number. For deciaml type constants, the scaling attribute is the number of decimal digits to the right of the decimal point.

Integer Attributes

An integer attribute can be written with an I' followed by the symbol you wish. An integer attribute is computed from length and scaling attributes and is thus also applicable only to a symbol which is the label of a statement defining fixed-point or floating-point constants (F, H, D, E, P, type Z, type K, and type G). A fixed-point integer attribute is equal to 8 times the length attribute, minus the scaling attribute, minus 1 (I'=8*L—S'—1). For floating-point, you obtain the integer attribute by subtracting 1 from the length attribute, multiplying by 2 and subtracting the scaling attribute (I'=2*(L'—1)—S').

A halfword fixed-point constant (H) would have a length attribute of 2 (L'=2) and a scale attribute specified as 4 (S'=4). Therefore the integer attribute would be (8x2)—8—1=7. A fullword fixed-point constant would have a length of 4 (L'=4) and a scale attribute specified here as 12 (S'=12). The integer attribute in this case would be (8x4)—12—1=19.

Since E is a floating-point fullword, its length attribute is 4 (L'=4). The scale attribute is specified to be 3 (S'=3). Thus the integer attribute is 2(4—1)—3=3. When we have a floating point doubleword constant (D), its length attribute is 8 (L'=8). The scale attribute is shown to be 6. The integer attribute we can then compute to be 2(8—1)—6=8. For decimal constants the integer attribute is the number of decimal digits to the left of the decimal point.

Count Attributes

You can use the count attribute of a call operand to reference the number of characters in the operand, excluding commas. This attribute is determined after substitution of any variable symbols, that is, it uses the replacement characters rather than the variable symbol to determine the count attribute. You can use the count attribute in SETA or DO operand fields, and in relational expressions of SETB and AIF operands that are within a macro.

If the operand selected is a sublist, the count attribute will include the parentheses and commas within the sublist.

Number Attributes

For call operands, you can also reference the number of operands in an operand sublist. You reference the number attribute by writing an N' followed by the symbol or parameter whose attribute you want. This number is equal to 1 plus the number of commas separating or indicating the ommission of operands in the sublist. This attribute is available in SETA, DO, SETB, or AIF directives.

If an operand is not a sublist, the number attribute is 1. If an operand is omitted, its value is 0.

Example:

```
LABEL     ΔOPERATIONΔ    OPERAND                  Δ              COMMENTS
1          10            16
          PROC     &PARAM,1
DATTR     NAME
*DISPLAY ATTRIBUTES OF MACRO INSTRUCTION OPERAND
.* THIS COMMENT IS NOT GENERATED
          LCLA     &SQ,&IO,&KQ,&NQ,&LQ
          LCLC     &TQ
&IO       SETA     I'&PARAM(1,1)
&SQ       SETA     S'&PARAM(1,1)
&KQ       SETA     K'&PARAM(1,1)
&NQ       SETA     N'&PARAM(1)
&LQ       SETA     L'&PARAM(1,1)
&TQ       SETC     T'&PARAM(1,1)
          DC       C'&PARAM(1,1)'  THIS IS THE OPERAND
          DC       Y(&LQ)          LENGTH ATTRIBUTE OF PARAM
          DC       Y(&KQ)          COUNT ATTRIBUTE OF PARAM
          DC       Y(&IO)          INTEGER ATTRIBUTE OF PARAM
          DC       Y(&SQ)          SCALE ATTRIBUTE OF PARAM
          DC       Y(&NQ)          NUMBER OF OPERANDS IN SUBLIST
          DC       C'&TQ'          TYPE ATTRIBUTE OF PARAM
          END
```

Glossary

A

absolute expression

An expression whose value is unchanged by program relocation. The absolute expression can be an absolute term or any combination of absolute terms. Arithmetic operators are permitted between absolute terms.

Examples of absolute terms are: a symbol that has an absolute value, a self-defining term, or a length attribute reference.

Relocatable terms alone or relocatable terms in combination with absolute terms can be contained within an absolute expression. This type of absolute expression requires that each relocatable term be paired with another relocatable term that has the opposite sign and the same relocatability attribute. The paired terms need not be contiguous.

The effect of relocation is canceled by the pairing of relocatable terms with the same relocatable attribute and opposite signs. The absolute expression is thereby reduced to a single absolute value.

The following are absolute expressions:

```
A
A+A—A
A—A+A+A
R+A—R
R—R+A
(R—R)*A
A*A
```

where:

A
: Is an absolute term.

R
: Is a relocatable term.

advance listing (EJECT)

Controlled by the EJECT directive.

arithmetic operators

The symbols +,—,*,/,//,*/. The intrinsic meanings of +,—,*, and / are the usual ones; that is, + indicates addition, — indicates subtraction, * indicates multiplication, and / indicates division.

The operator // denotes a covered quotient where A//B is equivalent to (A+B—1)/B. A covered quotient is equal to regular binary division except that if there is a remainder, a 1 is added to the regular quotient.

The operator */ denotes a binary shift left or right. A*/B indicates a left shift and is equivalent to A*2B. A*/(—B) indicates a right shift and is equivalent to A/2B.

C

character expression

A character string, a character substring, or a concatenation of strings or substrings. The maximum length of a character expression is 127 characters. Character expressions are used as operands of SET and SETC statements and as terms in a SETB relational expression.

A character string is at least one of the 256 valid characters enclosed by apostrophes. A character string, unlike a character self-defining term, is not converted and treated as a binary value. The value of a character string is determined by its length. Any character string is greater in value than any shorter character string. Rules for writing character strings are:

- Two apostrophes must be written within a character string to represent one apostrophe. The two apostrophes are replaced by a single apostrophe when the string is printed.
- Two ampersands must be written within a character string to represent one ampersand. Both ampersands are retained as part of the character string. A single ampersand within the character string is interpreted as the first character of a variable symbol.

A character substring is a valid character string followed by two arithmetic expressions separated by a comma and enclosed in parentheses. The format is:

character string (e_1,e_2)

where:

e_1
Specifies the leftmost character of the original character string to be included in the substring.

e_2
Specifies the number of characters to be in the substring.

The expressions e_1 and e_2 must be valid SETA expressions. If there are fewer characters (than the number specified by e_2) remaining after character number e_1 in the string, the resultant substring is shortened to include only valid characters of the original string. A null character string results if e_1 is greater than the number of characters in the original string.

character set

The overall character set of the assembler. This set is divided into the following classes:

Alphabetic set:

Alphabetic characters: the uppercase letters A through Z

Special letters: ? $ # @

Numeric characters: 0 through 9

Special characters : + — * / , = △ (blank) () . & ' > <

comments statement

A statement that, when written within a source code statement, causes the assembler to generate comments on the output listing. This type of comments statement is written with an asterisk in column 1 of the assembler coding form followed by the comment. To continue a comment on the following line, column 72 must contain X.

A special form of the comments statement is also available for use within macro definitions. This form is used to include comments in a macro definition that are not to be generated in the output listing. This type is written with a period in column 1 of the assembler coding form, followed by an asterisk (*) in column 2, followed by the comment.

Neither form of comments statement may be created by substitution for variable symbols. Substitution for variable symbols is not performed on comment lines.

Three statements are available for listing comments, error messages, or internal references. The PNOTE message statement may be used in either a macro definition or at the source code level. The MNOTE message statement may be used only in a macro definition. If either of these statements is generated by a macro definition, the statement will be printed, even if the NOGEN option of the PRINT statement is in effect. The comments statement may be used in macro definition form or in source code level form.

common storage definition

A common storage area for two or more separately assembled routines.

complex relocatable expressions

An expression that contains either 2 to 16 unpaired relocatable terms or a negative relocatable term in addition to any absolute or paired relocatable terms.

A complex relocatable expression may be written only in the operand field of either an A-type or Y-type address constant.

Some complex relocatable expressions are:

A—R
—R/I
A—R—R+R—R

where:

A
: Is an absolute term.

R
: Is a relocatable term.

concatenation
The joining together of:

- two character strings;
- two character substrings; or
- a character string and a character substring.

A period designates concatenation into a single string of characters. When a substring is to be concatenated with a following character string, the period may be omitted and concatenation is assumed.

conditional assembly
Statements used by the programmer to direct the assembler to:

- exclude lines of code from the assembler output;
- include a set of lines more than once in the assembly output; or
- establish and alter values to determine whether a set of lines should be included in the output listing.

Conditional assembly statements are used to control the pattern of coding generated within a macro definition and to define and assign values to set symbols that can be used to vary parts of generated statements.

conditional branch (AIF)
The statement that conditionally alters the sequence of source statement processing.

control section identification (CSECT)
The directive that indicates to the assembler the initiation or continuation of a control section.

D

define branch destination (ANOP)
The statement that facilitates branching by supplying a symbol in its label field.

define end of range (ENDO)
The statement used to indicate the end of the range of a DO statement.

define start of range (DO)
The statement that defines the starting point of the code and the number of times it is to be generated.

diagnostic listing
A listing of error statements. The diagnostic listing follows the assembly listing and contains a detailed accounting of any errors which occurred in the assembly. The listing contains the line number of the statement in which the error occurred, the error code, and a message indicating the cause of the error. The messages are listed in the order in which they occurred. A diagnostic listing is optional and can be suppressed by using the PARAM statement (3.22) with the LST=ND option in its operand field. The PARAM statement also provides the LST=DBG option for debugging a macro definition.

When a macro definition is retrieved from a library, the END statement is flagged if an error occurs during macro expansion. To obtain a diagnostic listing of the macro statement containing the error, you must use the LST=DBG option. If the macro definition is part of your source program, actual source statements are flagged if they contain errors. Each error is then listed in the diagnostic listing.

dummy control section identification (DSECT)

The directive that indicates to the assembler the areas defined in other modules.

E

expression

One or more terms connected by operators. A leading minus sign is allowed to produce the negative of the first term. Each term in the expression may be either a relocatable term or an absolute term. A term is absolute if its value is not changed by program relocation. A term is a relocatable term if its value is changed by program relocation. Two relocatable terms may be considered to be paired if they have opposite signs and have the same relocatibility attribute (that is, appear in the same control section).

Evaluation of expressions obeys the following rules:

- Multiplication and division of a relocatable term by an absolute 1 or multiplication of an absolute 1 by a relocatable term produces a relocatable term.
- Multiplication of any term by absolute 0 yields absolute 0 as a result.
- If a relocatable term enters any multiply or divide operation other than the above, an error flag is given and the result is treated as absolute.
- The number of unpaired relocatable terms at any point in the evaluation must not exceed 16.
- Intermediate results of the expression evaluation are full 32-bit values; however, the final result is the truncated rightmost 24 bits.

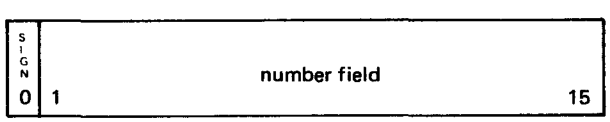
Three types of expressions — absolute, relocatable, and complex relocatable — obtain various characteristics from the term or terms that compose them.

F

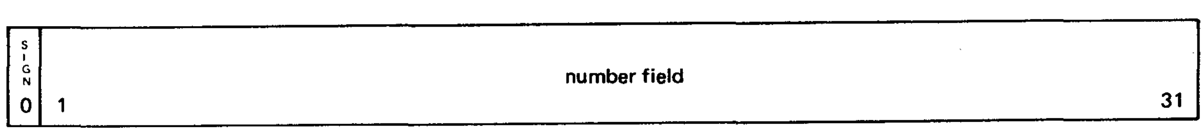
fixed-point number

A number represented in one of three fixed-length binary formats composed of a single positive or negative sign bit followed by a number field. When the sign bit is 0, the number represents a positive value; when 1, the number represents a negative value. Negative numbers are represented in twos complement notation, which is derived by inverting each bit of the binary number and adding 1 to the result of the inversion.

HALF WORD



FULL WORD



DOUBLE WORD

G

GBL
A general purpose global set symbol.

GBLA
An arithmetic global set symbol.

GBLB
A Boolean global set symbol.

GBLC
A character global set system.

generate literals (LTORG)
The directive that causes the assembler to generate literals previously defined.

H

high order
Leftmost data; most significant byte or bit.

I

include code from a library (COPY)
The directive that includes code into the source program.

input format control (ICTL)
The directive that specifies new values for the begin, end, and continuation columns.

input sequence control (ISEQ)
The directive that informs the assembler what columns contain the sequence information.

L

LCL
A general purpose local set symbol.

LCLA
An arithmetic local set symbol.

LCLB
A Boolean local set symbol.

LCLC
A character local set symbol.

leave blank lines on listing (SPACE)

The directive that causes the assembler to advance the paper in the printer.

length attribute of expressions

An attribute that is determined by the assembler and is a function of the leading term of the expression. If the first term of an expression is an absolute value, a length attribute of one byte is assigned to the expression. If the leading term is a symbol, the number of bytes attributed to the expression is the same as the length attributed to the symbol. Thus if TAG appears in the label field of an LH **(load half word)** instruction, it would have a length attribute of 4 since LH is a 4-byte instruction. In referencing the same label, the expression TAG+195 also has a length attribute of 4, but the expression 195+TAG has a length attribute of 1 because the leading term is a decimal self-defining term.

length attribute of symbols

The number of bytes assigned to the instruction, constant, or storage area involved. For example, the label of a 2-byte instruction has a length attribute of 2, and the label of a DS statement reserving 200 bytes would have a length attribute of 200. Symbols equated to location counter references or absolute value representations usually have a length attribute of 1. The duplication factor (constant or storage area) has no effect on the length attribute.

The maximum length attribute that can be generated by the assembler is 256 bytes; however, a DS may be used to reserve more than 256 bytes of storage.

The length attribute of a symbol may be referenced as a term in an expression by writing L' followed by the symbol. Thus if the symbol STOREND is the name of a full-word field,

```
L'STOREND
```

would be considered a term and would have a length of 4 bytes.

listing content control (PRINT)

The directive that controls the contents of the assembly listing.

literals

Terms that represent data in the source coding. The assembler replaces the literal with the address of the main storage location, in the literal table, of the value of the original literal. In the following example, the literal =C'AA' will be replaced in this instruction by the address of a 2-byte area in the literal table containing the binary value 11000001 11000001.

```
MOVEAA     MVC     TESTSW,=C'AA'
```

When the assembler recognizes a literal in the source code, it searches the table of literals that have been previously encountered. If a duplicate is found, then the relocatable address of the literal in the table replaces the original literal in the source code. If a duplicate is not found, then the value of the original literal is entered into the table and its address replaces the source code specification. Literals are similar in form to the operands of DC and DS statements.

A literal may be used in any machine instruction that specifies a storage address, except that the literal may not be specified as the receiving field operand of an instruction that modifies storage, i.e., a literal may be used only as the last operand of an application instruction. Literals may not be specified in address constants, shift instructions, or I/O instructions. Literals must always appear as the complete operand specification. They cannot be combined with other terms, nor with an explicit base register specification.

location counter reference

A reference maintained by the assembler for each control section created by the programmer. Each counter contains the next available location for the associated control section. After the assembler processes an instruction or constant, it adds the length of the instruction or constant processed to the correct location counter. The maximum value that the location counter can achieve is $2^{23}-1$.

Each instruction must have an address that is a multiple of two bytes. This type of address is said to fall on a half-word boundary. If the value of the location counter is not a multiple of 2 when assembling such an instruction, a 1 is added to the location counter before assigning an address to the current statement. Storage locations reserved in this way receive binary 0's when the program is loaded. Certain constants must be aligned to a half-word, full-word, or double-word boundary. Again the location counter is adjusted to the boundary, and the storage locations that were bypassed receive binary 0's when the program is loaded, unless the adjustment occurred as a result of a DS or ORG directive.

The current value of the location counter, under which the program is currently being assembled, is available for reference by the programmer. It is represented by the special character* (asterisk). If the asterisk is written as a term in an address constant or in an instruction operand expression, this character is replaced by the storage address of the leftmost byte allocated to that instruction or constant. All such implied references must be specified appropriately, since the asterisk (*) is also used as an arithmetic operator to indicate multiplication.

logical operators

The symbols **, ++, and ——. The characters ** represent the logical product (AND), the characters ++ represent the logical sum (OR), and the characters —— represent the symmetric difference, exclusive or (XOR).

Each bit of the first term is compared with its corresponding bit in the second term, and the result of the comparison is placed in the corresponding position in the resulting term. The result of the bit comparison for each operator is:

AND	
A**B	Result
1 1	1
1 0	0
0 1	0
0 0	0

OR	
A++B	Result
1 1	1
1 0	1
0 1	1
0 0	0

XOR	
A--B	Result
1 1	0
1 0	1
0 1	1
0 0	0

low order

Rightmost data; least significant byte or bit.

LSB

Least significant bit or byte, rightmost.

M

macro definition

A formalized pattern of code written once if a certain series of instructions (e.g., a routine) is needed more than once in a program or associated programs. The macro definition may be stored in a library for later use or submitted for assembly with the source code deck.

Macro definitions may be prepared in one of two separate formats: macro or proc. The elements of the macro and proc format types may not be mixed within a macro definition; however, macro definitions of both types are permitted within a program. Macro definitions contained in the source program may be preceded only by comment statements and the following assembler directives: ICTL, ISEQ, TITLE, SPACE, EJECT, and PRINT. Any of these directives except ICTL may appear between macro definitions. A macro definition within a macro definition (nesting) is not permitted in either the macro or the proc format.

model statements

The statements in a macro definition from which machine and assembler instructions are generated. Model statements contain from one to four entries, as follows:

- The label field may contain a symbol, a variable symbol, or a sequence symbol, depending on the operation defined. Comment statements may not be created by substitution for variable symbols.

- The operation field may contain any machine, assembler, or macro instruction mnemonic code except END, ICTL, ISEQ, or PRINT.

- Either ordinary symbols or variable symbols may be written in the operand field. The size of this field may not exceed 240 characters after substitution.

- The comments field may contain any combination of characters; however, substitution for variable symbols is not performed on this field by the assembler. Comments are written in the format of the statement the model represents.

- A macro instruction that is a model statement within a macro definition is called an inner macro instruction, while a macro instruction in the source module is called an outer macro instruction. A macro instruction that appears in a macro definition corresponding to an outer macro instruction is called a second-level macro instruction. A macro instruction that appears in the macro definition corresponds to a second-level macro instruction. Macro instructions within macro definitions are nested. The number of levels to which macro instructions may be nested in an assembly depends upon the amount of main storage available to the assembler.

- Because COPY statements within a macro definition are processed prior to the generation of code from a macro definition, they are not considered to be model statements nor are they ever processed as such.

- Model statements within a macro definition in proc format obey the same rules as model statements in macro format.

MSB

Most significant bit or byte, leftmost.

O

operators

The 12 mathematical functions in the assembler that designate the method, and implicitly the sequence, to be employed in combining terms or expressions. Evaluation of an expression begins with the substitution of values for each term. The operations are then performed from left to right in hierarchical order. The operation with the highest hierarchy number is performed first; operations with the same hierarchy number are performed from left to right.

Parentheses may be used to alter the order of evaluation. Multiplication by 0 equeals 0. The 12 operators are divided into three classes: arithmetic operators, logical operators, and relational operators.

P

privileged instructions

Instructions used by the operating system when the processor is in the supervisor state. If an application program (user program) attempts to execute a privileged instruction, a program exception interrupt will occur because the processor will be in the problem state. The following are the privileged instructions for the SPERRY UNIVAC Operating System/3 (OS/3).

- *Diagnose* (DIAG)
- *Halt and proceed* (HPR)
- *Insert storage key* (ISK)
- *Load control storage* (LCS)
- *Load program status word* (LPSW)
- *Start I/O* (SIO)
- *Supervisor load multiple* (SLM)
- *SOFTSCOPE forward scan* (SSFS)
- *Set storage key* (SSK)
- *Set system mask* (SSM)
- *SOFTSCOPE reverse scan* (SSRS)
- *Supervisor store multiple* (SSTM)
- *Service timer register* (STR)

program status word (PSW)

A special register containing information on the status of the program being run. The PSW contains the condition code, interrupt code, and the address of the next executable instruction. *See status switching instructions.*

PSW

See program status word.

R

relational operators

The equals symbol (=), the greater than symbol (>), and the less than symbol (<).

The equals operator is used to compare the value of two terms or expressions. If the two values are equal, the assembler assigns a value of 1 to the expression; otherwise, a value of 0 is assigned.

The greater than operator makes a comparison between two terms or expressions. If the value of the first (left) term is greater than the value of the second (right) term, then a value of 1 is assigned to the expression; otherwise, a value of 0 is assigned.

The less than operator compares the value of the first (left) expression or term with the second (right) expression. If the value of the first expression is less than the value of the second one, then a value of 1 is assigned to the expression; otherwise, a value of 0 is assigned.

For the expression A+B>C, if the expression A+B has a value greater than a value of C, then the assembler assigns a value of 1 to the expression; otherwise, a value of 0 is assigned.

A relational expression consists of a relational operator and its two operands. The operands in a relational expression may be either two character expressions or two arithmetic expressions. A character expression may not be compared to an arithmetic expression. Character expressions are valid only on conditional assembly directives.

Since the evaluation of a relational expression yields an arithmetic result, a relational expression may be used as a term in an arithmetic expression.

relocatability attributes

Values that are assigned to symbols defined in the label field of a source code line representing an instruction, constant, or storage definition. A relocatable symbol is a symbol whose address would change by a given number of bytes if the program in which it appears is relocated the same number of bytes from its originally assigned address. Relocatable symbols are assigned values relative to the location counter. Decimal, character, binary, and hexadecimal representations are all absolute terms and have a relocation attribute of 0.

relocatable expressions

An expression whose value changes with program relocation. All relocatable expressions must be positive values.

Relocatable terms alone or relocatable terms in combination with absolute terms can be contained within a relocatable expression.

Either type of relocatable expression requires the following conditions:

- All but one relocatable term must be paired.
- A minus sign must not precede the unpaired (remaining) relocatable term.

- Each pair of relocatable terms must have opposite signs and the same relocatability attribute.
- The paired relocatable terms do not have to be contiguous.

Using the above requirements, a relocatable expression is thereby reduced to a single relocatable term. The following are relocatable expressions:

R
R/I
R+A or A+R
R—R+R
R—A
R*I or I*R

where:

A
Is an absolute term.

R
Is a relocatable term.

reproduce following record (REPRO)

The directive used to reproduce a record in the assembler output.

S

SDT

See self-defining terms.

self-defining terms (SDT)

Terms that represent fixed values. They are presented by the programmer in a form that is easily recognized and its value is understood without the need for computation. SDTs are not relocatable; they can be used to specify immediate data, registers, addresses, and masks. They can be used in assembler directives as well as in application instructions and can be part of an expression. The size of an SDT depends on where it is used. When used to designate a register, it cannot exceed a value of 15. After conversion by the assembler to a binary format, the value is right-justified and filled with binary zeros on the left to fit the designated field. SDTs can be represented in binary, hexadecimal, decimal, or character form.

When a 24-bit hexadecimal, binary, or character SDT has a 1 in the sign bit position, the SDT will be treated as a negative term in the evaluation of an arithmetic expression.

- A binary SDT consists of a series of 24 zeros and ones enclosed in apostrophes and preceded by the letter B (e.g., B'101',B'11110000',B'00101'). The field is filled with high order zeros when necessary.
- A hexadecimal SDT consists of up to six hexadecimal digits enclosed in apostrophes and preceded by the letter X (e.g., X'F0',X'C1',X'F1F0F0'). Each hexadecimal digit represents a half byte of information.

- A decimal SDT is an unsigned decimal number consisting of up to eight digits having a value of 0 through 16,777,215 (2^{24}—1) (e.g., 0, 32, 16000000). This number is converted by the assembler to a binary value occupying one, two, or three bytes.

- A character SDT consists of up to 3 characters of the 256 valid characters of which only 63 are printable. The characters must be enclosed in apostrophes and preceded by the letter C (e.g., C'A', C'ABC', C'123', C'A1'). Each ampersand or apostrophe to be included in a character representation must be indicated by a double ampersand or double apostrophe respectively. In this case, there may be more than three characters within the apostrophes that delimit the SDT (e.g., C'3''S' produces 3'S; C'A&&B' produces A&B).

set symbol

A type of variable symbol. The rules for writing set symbols are the same as for other variable symbols:

- An ampersand (&) is followed by an alphabetic character followed by up to six additional characters (total maximum characters: eight)

- If the ampersand is omitted, the assembler interprets the character string as a symbol and not as a set symbol.

Because set symbols are evaluated in the macro generation phase of the assembler, they may be used as counters, switches, or values to control the sequence of code generated. Unlike an ordinary symbol, the value assigned to a set symbol may be altered during assembly. A set symbol may be either global or local. A global set symbol, once declared and given a value by a SET statement, retains the same value until that value is changed by another SET statement. A local set symbol is defined only within the macro definition in which it is declared. The value of a local set symbol within one macro definition is not affected by the declaration of either a global or local set symbol with the same name in another macro definition.

Do not use &SYS as the first four characters of any symbol because they are reserved for the use of system variable symbols.

Set symbols must be declared after macro prototype or NAME statements and before being referenced.

Four statements are provided to assign values to set symbols: SETA, SETB, SETC, and SET. The statement used depends on the statement chosen to declare the set symbol.

- SETA

  Assigns values to set symbols declared in either LCLA or GBLA.

- SETB

  Assigns values to set symbols declared in either LCLB or GBLB.

- SETC

  Assigns values to set symbols declared in either LCLC or GBLC.

- SET

  Assigns values to set symbols declared in either LCL or GBL.

special characters

The 14 special characters that are not part of the alphabetic set, are not special letters, and are not numerals. The special characters with their hexadecimal codes are:

Special Character	Hexadecimal (EBCDIC) Code	Special Character	Hexadecimal (EBCDIC) Code
+	4E	(left parenthesis	4D
- (minus)	60	) right parenthesis	5D
*	5C	. (period)	4B
/	61	&	50
, (comma)	6B	' (prime)	7D
=	7E	>	6E
△ (blank)	40	<	4C

special letters

The four special letters are:

Special Letters	Hexadecimal (EBCDIC) Code
?	6F
$	5B
#	7B
@	7C

specify location counter (ORG)

The directive that sets or resets the location counter to a specified value.

status switching instructions

The instructions that provide the capability of altering processor operating characteristics. The *set program mask* (SPM) and *supervisor call* (SVC) instructions replace part of the current program status word (PSW).

The format of the PSW is:

SYSTEM MASK								KEY	MODE							INTERRUPT CODE
T	IOST	SPARE	SPARE	SPARE	SPARE	SPARE	SPARE		A	PR	PS	SPARE	M	MON		
0	1	2	3	4	5	6	7	8 11	12	13	14	15	16 18	19	20 23	24 31

PROGRAM MASK						INSTRUCTION ADDRESS
ILC	CC	B	D	E	S	
32 33	34 35	36	37	38	39	40 63

For information on the format, description, and use of the PSW, see the processor programmer reference, UP-8052 (current version).

The *test and set* (TS) instruction is used to control a byte in main storage to act as an indicator.

symbols

Identifications appearing in the label field of a statement defining an instruction, constant, or storage area that are assigned the address value of the first byte of the source statement with which the symbol is associated. The following rules apply to the use of symbols used as labels.

- Must start in column 1
- Must start with an alphabetic character or special letter
- Must consist of only alphabetic characters, numeric characters, and special letters.
- Must not be longer than eight characters.
- Must not include a space (blank) or other special character
- Must be followed by a blank

The assembler associates three attributes with each symbol it processes. These attributes are value, length, and relocatability. Symbols defined by the EQU directive adopt the attributes of the expression in the operand field of the statement.

Once symbols are defined in the label field, they can be used as operands to represent the value which was defined.

T

terms

Values coded by the programmer or computed by the assembler. There are five classes of terms recognized by the assembler.

- Self-defining terms (SDT)
- Literals
- Symbols
- Location counter references
- Length attribute references

Self-defining terms are fixed values the programmer codes, such as 33,P'591',X'OF',B'11100110', or C'EBW'. Literals can have their value specified by the programmer or computed by the assembler and could look like =X'FO',=C'A', =P'—1', or =B'00001000' as used in storage-to-storage instructions (e.g., CLC TAGA,=C'A'). Symbols, location counter references, and length attribute references are assigned values by the assembler.

U

unassign base register (DROP)

The directive that informs the assembler specified registers are no longer available for base register assignment.

unconditional branch (AGO)

The statement that unconditionally alters the sequence of source statement processing.

V

value attribute

The value assigned a symbol when it appears in the label field of any source code statement other than a comment. A symbol appearing in the label field of an EQU or ORG directive is assigned the value of the expression in the operand field. In all other cases, the value assigned is the current value of the location counter after the adjustment to a half-word, full-word, or double-word boundary, if necessary. The value is assigned to the current label before the location counter is incremented for the next instruction, constant, or storage definition. Thus, if a symbol appears in the label field of a statement defining an instruction, constant, or storage area, the symbol is assigned a value equal to the storage area address of that instruction, constant, or storage area.

The value of a symbol must lie in the range -2^{23} through $2^{23}-1$.

variable symbol

A symbol consisting of two to eight characters; the first is an ampersand (&), the second is a letter (A through Z) or a special character (? $ # @), and each of the remaining characters is a letter, special character, or digit (0 through 9).

A variable symbol may be:

- a symbolic parameter;
- a set symbol;
- the label of a DO statement; or
- a system variable symbol.

Variable symbol symbolic parameters represent either the label or one of the operands of the macro instruction by which the macro definition was named.

The following rules apply to the use of variable symbols:

- A variable symbol may not be used to generate a new sequence symbol, a SET symbol, a parameter, or a system variable symbol.

- A variable symbol may not be used in the label or operand field of an END, ICTL, ISEQ, COPY, or PRINT directive.
- No variable symbol replacement is performed on the line following a REPRO directive.
- Variable symbol replacement must not produce leading blanks in the label or operand fields.

A variable symbol may appear in a statement concatenated (joined) with other variable symbols or characters. If a variable symbol is immediately followed by a letter, digit, left parenthesis, or period, a period must be written after the variable symbol to distinguish the variable symbol from the characters that follow it. The variable symbol and the period following it are replaced by the characters representing the value of the variable symbol. The period does not appear in the printed statement. If a period is between a character string (not in quotes) and a variable symbol (in that order), the period is considered part of the character string and will appear in the printed statement.

The period after the variable symbol is optional if the variable symbol terminates with a right parenthesis or is followed by another variable symbol or a special character other than a left parenthesis or a period.

SPERRY UNIVAC

Cut along line.

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

(Document Title)

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

(Name of User)

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD

CUT