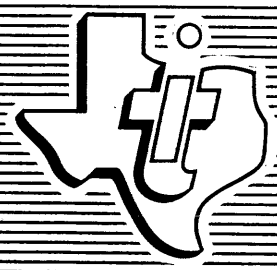
PROGRAMMER'S GUIDE TO PROCEDURE PROGRAMMING

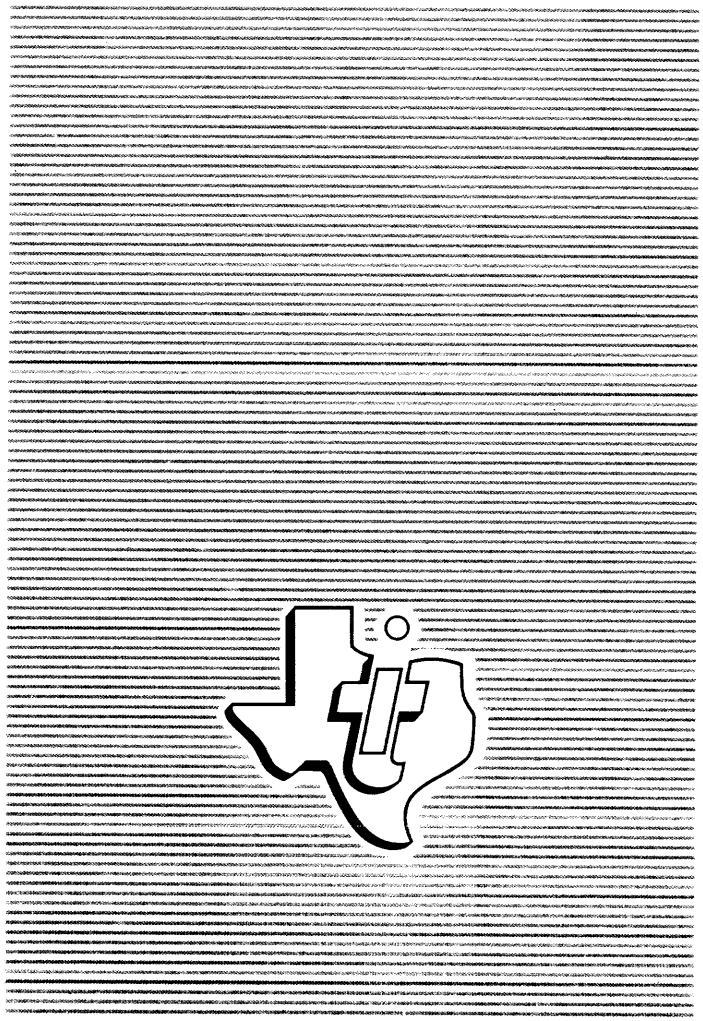


TEXAS INSTRUMENTS
INCORPORATED

ASC

PROGRAMMER'S GUIDE TO

PROCEDURE PROGRAMMING



Equipment Group
P. O. Box 2909
Austin, Texas 78767

TEXAS INSTRUMENTS
INCORPORATED

S1007P
June 1971

1158

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

TABLE OF CONTENTS (Continued)

SECTION I
GENERAL DESCRIPTION

1-1. THE ASC ASSEMBLER

The Assembler implemented for the ASC provides for symbolic coding of programs to be executed in either the Peripheral Processor or the Central Processor.

There are directives which are commands to the Assembler that define the machine characteristics of the machine for which object code is to be produced. These are used to define to the Assembler the differences between the two processors.

There are directives that inform the Assembler of conditions to be expected at assembly time, of conditions to be expected at object program execution time, and of the nature of symbols used by the programmer.

There are procedures built into the Assembler which translate data in a form convenient to the programmer into the object data formats usable by the machine.

Mnemonics for actual machine codes are the names of procedures that build the proper object code machine words.

The most powerful aspect of the Assembler is the directives which permit the programmer to define procedures by name for reference in coding segments by simple mnemonics.

1-2. CODING MEDIA

A source program is a sequence of source statements punched into cards and entered into the computer by a card reader.

1-3. PUNCHED CARD

The card format is a standard 80 column punched card.

1-4. CODING FORM

Source statements may be written on the standard coding form, shown in Figure 1-1. One line of code on the form is punched into one card; vertical columns on the form correspond to card columns.

Space is provided for program identification and for instructions to keypunch operators. The body of the coding form consists of the statement field, columns 1 through 72, and the identification sequence field, columns 73 through 80.

ASC CODING FORM

PROGRAM		PUNCHING INSTRUCTIONS	GRAPHIC									PAGE	OF
PROGRAMMER	DATE		PUNCH										

STATEMENT				IDENTIFICATION SEQUENCE
LABEL 8	COMMAND 10 15 17 20	OPERAND 25 30 35 40	REMARKS 45 50 55 60 65 71	73 80

Figure 1-1. Coding Form

SECTION II
LANGUAGE ELEMENTS

2-1. CHARACTER SET FOR THE ASC

The Assembler recognizes the EBCDIC character set as standard notation. That is, characters are interpreted as punched on the IBM 029 keypunch. References in this manual are made to alphabetic characters (A through Z), numeric characters (0 through 9), and special characters (all the rest).

All characters except the double quotation mark (") and the semicolon (;) may be used in character strings, and these also may be used freely in the remark field and in comments. The period (or decimal point), dollar sign ($), and question mark (?) may be used in symbols along with alphabetic and numeric characters. Most of the special characters have unique meanings to the Assembler.

The double quotation mark (") inside a character string will terminate the string. The semicolon (;) when used inside character strings will terminate the card image and the string will be continued on the next line beginning with the first non-blank character.

2-2. PRINTABLE CHARACTERS

Table 2-1 contains a list of the non-alphanumeric, printable characters and their names without regard to their special meanings to the Assembler.

2-3. SPECIAL CHARACTERS

Table 2-2 lists the special characters which have unique meaning to the Assembler.

2-4. ITEMS

An item consists of a combination of one or more characters. An item may be a symbol, decimal integer, character string, hexadecimal integer, location counter, floating point item, fixed point item, literal, or intrinsic function.

Table 2-1. Printable Characters

CHAR-ACTER	NAME	CARD CODE	CHAR-ACTER	NAME	CARD CODE
	blank	blank	-	hyphen, or minus sign	11
¢	cent sign	12-8-2	/	slash (virgule)	0-1
.	period	12-8-3	,	comma	0-8-3
<	less than	12-8-4	%	percent sign	0-8-4
(	left parenthesis	12-8-5	—	horizontal bar	0-8-5
+	plus sign	12-8-6	>	greater than	0-8-6
|	vertical bar	12-8-7	?	question mark	0-8-7
&	ampersand	12	↑	vertical arrow	8-1
!	exclamation point	11-8-2	:	colon	8-2
			#	number	8-3
$	dollar sign	11-8-3	@	at	8-4
*	asterisk	11-8-4	'	apostrophe	8-5
)	right parenthesis	11-8-5	=	equals	8-6
;	semicolon	11-8-6	"	quotation marks	8-7
¬	not sign	11-8-7			

Table 2-2. Special Characters

CHARACTER	MEANING	CHARACTER	MEANING
#	hexadecimal	)	right parenthesis
@	indirect addressing	¢	augment indicator
,	separator	>	greater than
$	location counter	;	continuation
*	multiply and comments	¬	not (one's complement)
.	period or decimal point	"	EBCDIC string indicator
<	less than	=	equals or literal indicator
-	subtract		
/	divide	(	left parenthesis
+	add	blank	separator or space

2-5. SYMBOL

A symbol is represented as a string of from one to eight EBCDIC characters, the first of which must be alphabetic. The remaining characters may be alphabetic, numeric, . , $, # , or any other special characters not used by the Assembler for unique purposes. (See Table 2-2 for characters having meaning to the Assembler.)

VALUE: The value of a symbol is a name assigned to another item or other value.

Examples: AABBCCDD

Q.J$P?

2-6. CHARACTER STRING

A character string is any string of characters surrounded by double quotation marks (not to be confused with two single quotation marks). Semicolons (;) or double quotation marks (") cannot be parts of a character string because they operate on the string. Character strings assigned to symbols as their values cannot exceed 28 characters in length. Other character strings are restricted to 256 characters.

VALUE: The value of a character string is the EBCDIC representation of the characters found between the quotation marks. Each character string is converted into an even multiple of four characters (32 bits). Strings which do not contain a multiple of four characters are filled to the right with blanks.

Example: "AB*C"

2-7. DECIMAL INTEGER

A decimal integer is a string of unsigned decimal digits (0 through 9).

VALUE: The value of a decimal integer is the 32-bit (binary representation) base 10 value of the string of digits.

Examples: 19

5440

2-8. HEXADECIMAL INTEGER

A hexadecimal integer is a string of unsigned hexadecimal digits (0 through F) preceded by a #. The maximum number of characters after the # is 16.

VALUE: The value of a hexadecimal integer is the 32 or 64-bit (binary representation) base 16 value of the string of digits.

Example: #3B8FE5

2-9. FLOATING POINT ITEM

A floating point item is a string of decimal digits with a decimal point and optionally followed by a decimal exponent. The exponent is written as the letter E or the letter D followed by an integer constant. The item may be positive, zero, or negative. If either the initial string of decimal digits or the integral exponent is unsigned, the Assembler assumes the respective part to be positive. If a decimal exponent is given, the decimal point is not required in the initial string of digits. The item may assume one of three forms:

1. A string of decimal digits with a decimal point and without an exponent. This form is assumed, by the Assembler, to be single precision representation.

2. A string of decimal digits, optionally with a decimal point, followed by the letter E and an integral decimal exponent. The E specifies single precision representation.

3. A string of decimal digits, optionally with a decimal point, followed by the letter D and an integral decimal exponent. The D specifies double precision representation.

For both single and double precision representation, the value of the exponent, n, has the range: $-64 \leq n \leq +63$. The range of values M, a floating point item, may have is: (1) in single precision (32-bit representation), $16^{-65} \leq M \leq (1 - 16^{-6}) \times 16^{63}$ and true zero; and (2) in double precision (64-bit representation), $16^{-65} \leq M \leq (1 - 16^{-14}) \times 16^{63}$ and true zero; or, approximately, $5.4 \times 10^{-79} \leq M \leq 7.2 \times 10^{75}$. The number of bits used in the representation of the fractional part of a

floating point item does not significantly affect its range of values, but affects the precision of the values that may be represented.

If the maximum exponent value is exceeded, a syntax error is returned; but, if the significance is exceeded, truncation of the least significant bits occurs and no error message is returned.

VALUE: The value of a floating point item is the 32 or 64-bit binary representation of the string of digits with eight bits reserved for the exponent and with the remaining 24 or 56 bits left for the fraction. The exponent is represented in excess 64 notation. The fraction is normalized in its area.

Examples: 5.321E+6
6D-26
5.3
-5.2E6
2.718

2-10. FIXED POINT DECIMAL ITEM

A fixed point decimal item is a string of decimal digits, which may have a decimal point, followed by (1) a B or a BB, and by (2) a binary scale factor. The item may be positive, zero, or negative. If either the initial string of decimal digits or the binary scale factor is unsigned, the Assembler assumes the respective part to be positive. A positive binary scale factor shifts the binary representation of the quantity to the left by the specified number of binary units, and a negative binary scale factor shifts the binary representation of the quantity to the right by the specified number of binary units. Any part of the decimal numeral which would result in a binary fraction, when converted to binary and scaled, will be truncated. A single B specifies single precision, and a double B (i.e., BB) specifies double precision.

The range of values of a fixed point item, F, is restricted to:
$-2,147,483,648 \leq F \leq +2,147,483,647$ (i.e., $-2^{31} \leq F \leq 2^{31} - 1$).

VALUE: The value of a fixed point decimal item is the 32 of 64-bit binary representation of the string of digits with the representation determined by converting integer and fraction portions of the string separately and placing the result in either 32 or 64 bits, as determined by the precision designator, B or BB, respectively.

Examples: 3.21B+5
6B2
3.21BB+5
6BB2

2-11. LOCATION COUNTER

The coding symbol for the value of the location counter is $.

VALUE: The value of $ is the 32-bit current value at assembly time of the location counter.

Example: $+6

2-12. LITERAL

A literal is a constant which is the relative location of the start of one or more words of data. A literal is expressed in the form of an equals sign followed by the data to be contained in the relative location (see Topic 2-19).

VALUE: The value of a literal is the location of a constant.

Examples: =A
=6
=A+6

2-13. INTRINSIC FUNCTION

An intrinsic function is an item used to produce substitution of another item, expression, or list in its place. See Topic 2-21.

VALUE: The value of an intrinsic function is the identity of the particular parameter operated on by the function or is the value assigned to the condition of the parameter operated on by the function.

Examples: T(RHO)
VP(1)
IP(2, 1, 1)
CP(I)

2-14. OPERATORS

Items may be combined using the special character operators defined in Table 2-3. The table also gives hierarchy numbers for determining the sequence in which the value of an expression is computed. Operations with higher hierarchies are performed before operations having lower hierarchies. Operations with the same hierarchy are performed from left to right.

2-15. OPERATOR TYPES

Each operator falls under two type classifications: every operator is either a unary operator or a binary operator; and every operator is either an arithmetic operator, a relational operator, or a logical operator. See Tables 2-3 and 2-4 for the operator symbols and their uses.

UNARY OPERATION: A unary operation is one that involves only one operand.

BINARY OPERATION: A binary operation is one that involves two operands.

ARITHMETIC OPERATION: An arithmetic operation is one that yields algebraic quantities.

RELATIONAL OPERATION: A relational operation is one that yields a "TRUE" or "FALSE" quantity; i.e., 1 or 0, respectively.

LOGICAL OPERATION: A logical operation is one that yields a Boolean quantity.

2-16. EXPRESSIONS

An expression is an item or a series of items connected by operators. The sequence of operations performed in evaluating an expression is determined by the hierarchy of the operators in the expression. The hierarchy of operators is shown in Table 2-3. Operations with higher hierarchy numbers are performed first; operations with the same hierarchy are performed from left to right.

Table 2-3. Operator Hierarchies and Descriptions

HIER-ARCHY	SYMBOL	TYPE	DESCRIPTION
7	+	Unary Arithmetic	Plus
7	-	Unary Arithmetic	Minus (two's complement)
7	¬	Unary Arithmetic	Not (one's complement)
6	//	Binary Logical	Logical Binary Operator
5	*	Binary Arithmetic	Arithmetic Product
5	/	Binary Arithmetic	Arithmetic Quotient
4	+	Binary Arithmetic	Arithmetic Sum
4	-	Binary Arithmetic	Arithmetic Difference
3	<	Binary Relational	Arithmetic Less Than
3	¬<	Binary Relational	Not Less Than
3	=	Binary Relational	Arithmetic Equals
3	¬=	Binary Relational	Not Equals
3	<=	Binary Relational	Less Than or Equal
3	>	Binary Relational	Arithmetic Greater Than
3	¬>	Binary Relational	Not Greater Than
3	>=	Binary Relational	Greater Than or Equal
2	**	Binary Logical	Logical Product (AND)
1	++	Binary Logical	Logical Sum (OR)
1	--	Binary Logical	Logical Difference (Exclusive OR)
1	==	Binary Logical	Logical Equivalence

The length of an expression is limited by the number of continuation lines over which the statement may extend. The value of an arithmetic expression, E, is restricted to the range: $-2,147,483,648 \leq E \leq +2,147,483,647$ ($-2^{31} \leq E \leq 2^{31} - 1$). The value of an expression, E, containing an external symbol or symbols is restricted to the range: $-8,388,608 \leq E \leq +8,388,617$, ($-2^{23} \leq E \leq 2^{23} - 1$).

Floating point numbers are not valid in expressions which contain more than one item. That is, floating point arithmetic will not be performed at assembly

Table 2-4. Use of Operators

SYMBOL	GENERAL FORM	WHERE	RESULTS
+	+a	a is an algebraic expression	a
-	-a	a is an algebraic expression	two's complement of a
¬	¬a	a is an algebraic or logical expression	one's complement of a
//	a//i	a is a logical expression; i is an integer expression	shift a left i binary digits if i is positive; shift a right i binary digits if i is negative
*	a*b	a and b are algebraic expressions	the product of a and b
/	a/b	the numerator a is an algebraic expression; the denominator b is an algebraic expression	the quotient of a divided by b
+	a+b	a and b are algebraic expressions	the sum of a and b
-	a-b	a and b are algebraic expressions	the difference of a and b
<	a<b	a and b are algebraic expressions	true if a is less than b
¬<	a¬<b	a and b are algebraic expressions	true if a is not less than b
=	a=b	a and b are algebraic expressions	true if a is equal to b
¬=	a¬=b	a and b are algebraic expressions	true if a is not equal to b
<=	a<=b	a and b are algebraic expressions	true if a is less than or equal to b
>	a>b	a and b are algebraic expressions	true if a is greater than b
¬>	a¬>b	a and b are algebraic expressions	true if a is not greater than b
>=	a>=b	a and b are algebraic expressions	true if a is greater than or equal to b
**	a**b	a and b are logical expressions	logical product of a and b (AND)
++	a++b	a and b are logical expressions	logical sum of a and b (OR)
--	a--b	a and b are logical expressions	logical difference of a and b (exclusive OR)
==	a==b	a and b are logical expressions	logical equivalence of a and b

time. The Assembler will denote as an error any attempts to do arithmetic operations on double length floating point numbers in expressions or on character strings longer than four characters.

Certain logical operations (**, ++, --, and ==) and all relational operations may be performed on values that require more than four characters (32 bits) to represent them.

2-17. SUBEXPRESSIONS

An expression may contain subexpressions, and subexpressions may contain other subexpressions. A subexpression is an expression enclosed in parentheses and it may appear wherever an item is valid. Subexpressions are evaluated before other items in an expression, and the innermost subexpression is evaluated first.

The value of an item or expression is right-justified in its generated result field, and unspecified leading bit positions will contain zeros; character strings are left-justified with blanks filled to the right in the last word for unjustified characters.

The value of the part of the expression or subexpression containing and affected by a relational operator (e.g., >, <, or =) is equated to one if the relation is true and equated to zero if the relation is false. For example, if E is an expression of the form:

X >Y

then, E is evaluated as a one (1) if the relation is true, or zero (0) if the relation is false. Also, if the assigned section of expression X is not the same as the assigned section of expression Y, then the expression E cannot be completed and is evaluated as false (zero).

Examples:

1. The following expression is evaluated as zero if R is a relocatable item:

   R-4 >37

2. The following expression is evaluated as zero if the subexpression (X >Y) is false and equal to A if the subexpression (X >Y) is true:

   A*(X >Y)

2-18. ASSUMED PARENTHESES

The following examples denote how parentheses are assumed, the results being governed by the hierarchies in Table 2-3.

Expression: -A//-I*2

Method:
1. Two's complement A
2. Two's complement I
3. Shift two's complement of A by value of two's complement of I
4. Multiply result by two

Assumes: ((-A)//(-I))*2

Expression: -A//(-I*2)

Method:
1. Two's complement I
2. Multiply result of two's complement of I by two
3. Two's complement A
4. Shift result of two's complement of A by result obtained in step 2

Assumes: (-A)//((-I)*2)

Expression: -A//-(I*2)

Method:
1. Multiply I by two
2. Two's complement A
3. Two's complement the result of I multiplied by two
4. Shift the result of the two's complement of A by the result obtained in step 3.

Assumes: (-A)//(-(I*2))

2-19. LITERALS

A literal is a constant. The constant is the relative location of the start of one or more words of data. The relative location is reserved by the Assembler, and the contents of the location are set to the value of the expression which specifies the data. An expression which is to be a literal is identified by being preceded by an equal sign (=). The Assembler reserves sufficient contiguous words to contain the

value of the expression. The number of words reserved for expressions which do not contain forward references is determined by the number of bit positions required to specify the value. Expressions that contain forward references are assumed to require no more than one word to specify their respective values.

Literals which have the same value are stored only once, whenever possible. Reaching the end of an assembly or using the LITORG directive (see Topic 4-13) causes all literals identified since the last LITORG directive or since the start of the assembly to be assigned locations and to be output. Literals appearing after a LITORG directive that are duplicates of values occurring before that LITORG directive will be assigned at least two separate locations. Further duplication will occur if the expression composing the literal is not a single item and all of the quantities composing the expression are not defined prior to their appearance in the expression.

Subexpressions and lists will not be made into literals. The value of expressions that identify literals is restricted to 28 characters in length. Multiple-word literal values will be assigned locations beginning on even-word boundaries (locations that are multiples of two). The initial literal location assignment occurring after a LITORG directive or at the end of an assembly will always start at an even-word boundary. Words that are skipped to achieve even-word alignment will not be cleared. The literal table is adjusted so that multiple-word literals are output first. Address constants will be placed in the literal table when symbols with relocatable values are used as literals.

Examples:

LITERAL	VALUE
=A	Address of a word that contains the address of the symbol A.
=6	Address of a word that contains the value 6.
=A+6	Address of a word that contains the value of the expression A+6. If this literal is used before A is defined, more than one constant with this value will be allocated.
=A+1=B	Address of a constant that contains 0 or 1 (the value of the expression A+1=B).
A+=B	Error
=(A, B, . . .)	Error

2-20. LISTS

A list is a set of items, expressions, or sublists separated by commas. In the most trivial case a list may be a single item. A sublist is a list enclosed in parentheses. Lists are used in the operand field of a statement.

If a list of parameters is enclosed by a single set of parentheses, these parameters are considered to be second level parameters. In the list

A, (B, C), D

B and C are second level parameters while A and D are first level. Parameter 2 (at first level) is a sublist.

Restrictions: The maximum number of expressions in a list at one level is 15.

The maximum number of levels of parentheses in a list is five.

The value of a parameter which is nonexistent or uncoded is always zero; e.g., for a general list expa, (expd, expb), expx that is coded A1, (A4,), the expb and expx would both be evaluated as zero.

2-21. INTRINSIC FUNCTIONS

An intrinsic function is an operation performed on or applied to an expression or a list. Some intrinsic functions (global intrinsic functions) may be used outside or inside procedures, whereas others (local intrinsic functions) may be used only in procedures.

Intrinsic function usages may be nested.

Intrinsic functions may be used in the label field, the command field, or the operand field of an Assembler language statement.

2-22. GLOBAL INTRINSIC FUNCTIONS

A global intrinsic function is one that may be used either outside or inside procedures and is of the general form:

F(e)

where F is the intrinsic function name and (e) is an expression.

2-23. LOCAL INTRINSIC FUNCTIONS

A local intrinsic function is one that may be used only in procedure definitions (refer to Section VI) and is of the general form:

F(i, i, i, . . .)

where F is the intrinsic function name and (i, i, i, . . .) is the parameter index. The parameter index is a list of up to five expressions.

The local intrinsic functions fall into two categories:

1. those which refer to list parameters in the operand of the PROC statement or of the NAME statement in the procedure definition and have the format:

   XQ(x, y, . . .)

   where X is the function symbol and x, y, . . . are parameter indices, and

2. those which refer to list parameters in the operand of the statement that calls the procedure and have the format:

   XP(x, y, . . .)

   where X is the function and x, y, . . . are parameter indices. See Topic 2-20.

The P or Q is used in combination with one of the function symbols as follows: V,R, C, I, A, L, or N.

Note: The names representing intrinsic functions, e.g., VP, IP, are not reserved symbols; i.e., these symbols may be defined for other uses by the programmer.

2-24. Parameter Indices

The expressions enclosed in parentheses following the function symbol are parameter indices. Each index is the positional number of the parameter within the level. Levels are specified by the sequential position of the number, not by the number itself; e.g., (x, y, z) would specify the zth parameter of sublist y of list x.

An index of zero, (0), refers to the entire list. A parameter index of the form (m, 0) is the same as the parameter index (m).

Note: The parameter indices may be coded symbolically so long as the item or expression is evaluated as an absolute value.

Example: In the list

A, B, (C, D, (E, F), G), H

the reference indices for the various parameters are:

PARAMETER	INDEX	PARAMETER	INDEX
A, B, (C, D, (E, F), G), H	(0)	D	(3, 2)
A	(1)	(E, F)	(3, 3)
B	(2)	G	(3, 4)
(C, D, (E, F), G)	(3)	E	(3, 3, 1)
H	(4)	F	(3, 3, 2)
C	(3, 1)		

Limits and Restrictions: Since the maximum number of items in a list at one level is 15 and the maximum number of levels of parentheses is 5, the largest parameter index is 15 and the maximum number of parameter indices in an index is 5. Thus, the index, (15, 15, 15, 15, 15), is the maximum value of an intrinsic reference.

In nesting intrinsic functions wherein one of the intrinsic functions transfers a list of parameters rather than a single item, that intrinsic function must be the last parameter of the intrinsic function list; e.g., in the list

RP(CP(1, 1), 1, CP(1, 3))

the expression is legal if CP(1, 1) is an item and illegal if it is a list; CP(1, 3) may be either an item or a list.

If an intrinsic function is used in the command field, it cannot refer to a directive name.

The value of a parameter which is nonexistent is always zero; e.g., for a general list expa, (expd, expb), expx that is coded A1, (A4,), expb and expx would both be evaluated as zero.

2-25. REFERENCING FUNCTIONS

A referencing intrinsic function is one that accesses a particular item, expression, or list and that inserts the identity of a parameter, copies a parameter, or merges a parameter into the current operand list. The referencing functions are:

SYMBOL	MEANING
VP or VQ	Insert identity of parameter in current operand list
RP or RQ	Merge parameter into current operand list
CP or CQ	Copy parameter into current operand list

2-26. Insert Identity Function - VQ(n) or VP(n)

An insert identity intrinsic function, VQ(n) or VP(n), denotes that the identity of the parameter specified by the parameter index, n, is to be inserted in the current operand list, i.e., the operand list in which the intrinsic function appears. The parameter index, n, may be an absolute expression or a list composed of absolute elements, e.g., VQ(X>Y), VP(2, 1), or VP(2, X>Y).

The primary use of the insert identity function is to access the identity of a parameter which is a symbol, a value, or an expression and not a list.

According to the characteristics of the parameter specified by n, its identity is as follows:

PARAMETER	VALUE
expression	expression
list	zero
nonexistent	zero
@, =, or ¢	removed

Example: Given, in a procedure calling statement, the list

=A, ((@B, C), D*(E>F)), =(G)

the following intrinsic functions, in a current statement, are:

FUNCTION	REPLACED BY
VP(0)	0 (reference index zero refers to entire list)
VP(1)	the address of the literal, A
VP(2)	0 (parameter two is a list)
VP(3)	0 (flagged as an error; a list, even of one item, cannot be a literal)
VP(4)	0 (parameter four is nonexistent)
VP(2, 1)	0 (parameter 2, 1 is a list)
VP(2, 2)	D, or zero depending upon whether the value of E is greater than the value of F
VP(2, 1, 1)	B
VP(2, 1, 2)	C

2-27. Copy Parameter Function - CQ(n) or CP(n)

A copy parameter function, CQ(n) or CP(n), denotes that the parameter specified by the parameter index, n, is to be copied into the current operand list. The parameter index, n, may be an absolute expression or a list composed of absolute elements, e.g., CP(X>Y) or CQ(1, 2, 3).

If the parameter index, n, is zero, e.g., CP(0), the entire copied list will be enclosed in parentheses.

Literals are replaced by their addresses, but the literal sign can still be detected by the LQ or LP intrinsic function in a procedure called that uses this function as an operand of the call.

The copy parameter function is replaced by:

PARAMETER	IDENTITY
if not a list	exact copy
if a list	exact copy

Example: Given, in a procedure calling statement, the list

=A, ((@B, C), D*(E>F)), (G)

the following intrinsic functions, in a current statement, are:

FUNCTION	REPLACED BY
CP(0)	the list (address of literal A, ((@B, C), D*(E F)), (G))
CP(1)	address of literal A
CP(2)	((@B, C), D*(E F))
CP(3)	(G)
CP(2, 1)	(@B, C)
CP(2, 2)	D, or zero depending upon whether the value of E is greater than the value of F
CP(2, 1, 1)	@B
CP(2, 1, 2)	C
CP(3, 1)	G

2-28. Merge Parameter Function - RQ(n) or RP(n)

A merge parameter function, RQ(n) or RP(n), denotes that the parameter specified by the parameter index, n, is to be merged into the current operand list. The parameter index, n, may be an absolute expression or a list composed of absolute elements, e.g., RP(MU=NU) or RQ(2, 1, 1).

The merge parameter function is replaced by:

PARAMETER	IDENTITY
if not a list	identity of the parameter
if a list	entire list with outer level parentheses and outer special characters removed

The primary use of the merge parameter function is to merge parameters from the referenced list into the current list; e.g., given the list

A, (R, N, X), LAMBDA

in a procedure calling statement, and, in the current statement, the list

PI, RP(2), RHO

the parameters in the current statement would have the following parameter indices:

PARAMETER	PARAMETER INDEX
PI	(1)
R	(2)
N	(3)
X	(4)
RHO	(5)

as though the current list were of the form:

PI, R, N, X, RHO

If the parameter index, n, is zero, an assumed pair of parentheses is removed before substitution; e.g., the list

@(A, B)

is assumed to be

(@(A, B))

so that the @ will be retained in the replacement.

Example: Given, in a procedure calling statement, the list

=A, ((@B, C), D*(E>F)), (G)

the following intrinsic functions, in a current statement, are:

FUNCTION	REPLACED BY
RP(0)	the address of the literal A, ((@B, C), D*(E>F)), (G)
RP(1)	the address of the literal A
RP(2)	(@B, C), D*(E>F)
RP(3)	G
RP(4)	0 (parameter four is nonexistent)
RP(2, 1)	@B, C
RP(2, 2)	D, or zero depending upon whether the value of E is greater than the value of F
RP(2, 1, 1)	B
RP(2, 1, 2)	C
RP(3, 1)	G

Note: Note that an expression preceded by an = is converted to a literal reference independently of list processing.

2-29. Summary of Referencing Functions

A summary of the referencing intrinsic functions is as follows:

INTRINSIC FUNCTION	IF NOT A LIST	IF A LIST
VQ(n) or VP(n)	Surrounding characters removed, substitute identity	Value of zero
RQ(n) or RP(n)	Surrounding characters removed, substitute identity	Remove surrounding parentheses and everything outside of them, substitute list
CQ(n) or CP(n)	Copied as is	Copied as is

2-30. ATTRIBUTE FUNCTIONS

An attribute of a parameter is the characteristic, or the value of the characteristic, of the parameter; e.g., the fact that the parameter is a literal is a characteristic, or the value of the base of the parameter is the value of a characteristic.

An attribute function either determines the value of the characteristic of a specified parameter or determines on a true or false basis whether a specified parameter has a certain characteristic.

Some attribute functions are global and others are local. All global functions are also attribute functions.

2-31. Local Attribute Functions

A local attribute function is one that may be used only in a procedure definition and that detects indirect address flags, or augmentation flags, or counts the number of parameters in a list. The local attribute functions are:

SYMBOL	MEANING
IP or IQ	Detect indirect flag in parameter
AP or AQ	Detect augment flag in parameter
LP or LQ	Detect literal flag in parameter
NP or NQ	Count number of expressions in parameter

2-32. Detect Indirect Function - IQ(n) or IP(n)

The detect indirect function, IQ(n) or IP(n), is used to detect indirect addressing flags. If an @ precedes the expression specified by the parameter index, n, IQ(n) or IP(n) is replaced with a one; if not, the intrinsic function is replaced with zero.

The parameter index, n, may be an absolute expression, or it may be a list of absolute elements, e.g., IQ(A<=B) or IP(3, 1).

Note: An @ applies to an entire expression and, therefore, must precede it; i.e., TAU+@BETA is illegal.

Example: Given, in a procedure calling statement, the list

=A, ((@B, C), D*(E>F)), @(G)

the following intrinsic functions, in a current statement, are:

FUNCTION	REPLACED BY	FUNCTION	REPLACED BY
IP(0)	0	IP(2, 2)	0
IP(1)	0	IP(2, 1, 1)	1
IP(2)	0	IP(2, 1, 2)	0
IP(3)	0 (flagged as an error)	IP(3, 1)	0
IP(2, 1)	0		

2-33. Detect Literal Function - LQ(n) or LP(n)

The detect literal function, LQ(n) or LP(n), is used to detect literal flags. (See Topic 2-19). If an = precedes the expression specified by the parameter index, n, LQ(n) or LP(n) is replaced with a one; if not, the intrinsic function is replaced with zero.

The parameter index, n, may be an absolute expression, or it may be a list of absolute elements, e.g., LQ(1, 2) or LP(A>B).

Example: Given, in a procedure calling statement, the list

=A, ((@B, =C), D*(E>F)), =(G)

the following intrinsic functions, in a current statement are:

FUNCTION	REPLACED BY	FUNCTION	REPLACED BY
LP(0)	0	LP(2, 1)	0
LP(1)	1	LP(2, 2)	0
LP(2)	0	LP(2, 1, 1)	0
LP(3)	0 (flagged as an error, a list cannot be a literal)	LP(2, 1, 2)	1
		LP(3, 1)	0

2-34. Detect Augment Function - AQ(n) or AP(n)

The detect augment function, AQ(n) or AP(n), is used to detect augmentation flags. (See PROGRAMMER'S GUIDE TO THE PERIPHERAL PROCESSOR). If a ¢ precedes the expression specified by the parameter index, n, AQ(n) or AP(n) is replaced with a one; if not, the intrinsic function is replaced with zero.

The parameter index, n, may be an absolute expression, or it may be a list of absolute elements, e.g., AQ(D>=B) or AP(4, 2).

Note: A ¢ applies to an entire expression and, therefore, must precede it; i.e., PHI- ¢ MU is illegal.

Example: Given, in a NAME statement, the list

¢A, ((@B, ¢C), D*(E<F)), ¢(G)

the following intrinsic functions, in a current statement, are:

FUNCTION	REPLACED BY	FUNCTION	REPLACED BY
AQ(0)	0	AQ(2, 2)	0
AQ(1)	1	AQ(2, 1, 1)	0
AQ(2)	0	AQ(2, 1, 2)	1
AQ(3)	0 (flagged as an error)	AQ(3, 1)	0
AQ(2, 1)	0		

2-35. Count Expressions Function - NQ(n) or NP(n)

The count expressions function, NQ(n) or NP(n), is replaced by the number of parameters in the sublist specified by the parameter index, n.

If the parameter specified by n has only one parameter and that parameter is not a sublist (i.e., enclosed in parentheses), NQ(n) or NP(n) is replaced with a zero; if the specified parameter is a sublist with only one parameter, NQ(n) or NP(n) is replaced with a one.

The parameter index, n, may be an absolute expression, or it may be a list composed of absolute elements, e.g., NQ(A>B, C<D) or NP(1, 4).

Example: Given, in a procedure calling statement, the list

=A, ((@B, C), D*(E<F)), (G)

the following intrinsic functions, in a current statement, are:

FUNCTION	REPLACED BY	FUNCTION	REPLACED BY
NP(0)	3	NP(2, 2)	0
NP(1)	0	NP(2, 1, 1)	0
NP(2)	2	NP(2, 1, 2)	0
NP(3)	1	NP(3, 1)	0
NP(2, 1)	2		

2-36. Global Attribute Functions

A global attribute function is one that may be used either inside or outside procedure definitions and that determines the value of some characteristic of the specified parameter.

2-37. Base Function - B(α)

The base intrinsic function, B(α), is replaced by the number of the base which yields the smallest non-negative result (displacement) when the value of that base is subtracted from the value of the expression, α.

If two or more bases yield the same least result, the highest numbered base is selected to replace B(α). See PROGRAMMER'S GUIDE TO THE CENTRAL PROCESSOR.

Limits and Restrictions: If all bases yield a displacement greater than 4095, an error message is generated by the Assembler.

The referent α cannot be a list. The base of a parameter within a list may be obtained through use of the referencing intrinsic functions, e.g., B(VP(1, 3)).

The base function is replaced by:

IF (α) IS:	D (α) IS:
external reference	error, message generated by Assembler
absolute	zero
relocatable	absolute

Examples: B(ALPHA)
B(VP(3))
B(CP(VP(1), 2))

2-38. Displacement Function - D (α)

The displacement intrinsic function, D(α), is replaced by the displacement value of the expression, α.

The displacement value of expression α is the smallest non-negative difference between the value of the expression, α, and the values of the bases (if any) in the table of applicable bases. See PROGRAMMER'S GUIDE TO THE CENTRAL PROCESSOR.

If all bases have a value of zero, the displacement of expression α is the displacement of the expression relative to the beginning of the control section.

Limits and Restrictions: If a displacement is greater than 4095, an error message is generated by the Assembler.

The referent α cannot be a list. The displacement of a parameter within a list may be obtained through the use of the referencing intrinsic functions, e.g., D(VP(1)).

The displacement function is replaced by:

IF (α) IS:	D (α) IS:
external reference	error, message generated by Assembler
$	relocatable value of $
absolute	zero
relocatable	absolute

Examples: D(ALPHA)
D(VP(3))
D(CP(VP(1), 2))

2-39. Section Function - T (α)

The section intrinsic function is replaced by the section number of symbol α, if a section number is valid. See Topic 2-41.

Limits and Restrictions: If α is an external reference, T(α) is greater than 256 and the value of T(α) is the sum of 256 plus the external symbol number. See Topic 4-5.

If (α) is absolute, T(α) is replaced by zero. T(α) is replaced by an absolute value. If α is in a dummy section (see Topic 4-17), T(α) is replaced by the negative of the dummy section number.

When using an assembler directive that produces multiple code (e.g., the DO directive), the argument, α, of the intrinsic function cannot be a forward reference.

The referent α cannot be a list. The section number of a parameter within a list may be obtained through the use of the referencing intrinsic functions; e.g., T(VP(2, 1)).

Examples: T(BETA)
T(CP(2))
T(CP(VP(2), 1))

2-40. Location Counter

The location counter symbol, $, when processed during evaluation of expressions, causes the current relative location in the assembly of the procedure call to be inserted in place of the symbol. In this sense, it acts somewhat like an intrinsic function.

General Example: Given, in a procedure calling statement or in a PROC or NAME statement, the list

=A=B, ((@C, =D), (E, F*(G<H)), I),@(J), (K)

the intrinsic functions, in a current statement, are replaced as follows:

PARAMETER INDEX	VQ OR VP	IQ OR LP	LQ OR LP	NQ OR NP
(0)	0	0	0	4
(1)	the address of the literal 1 or the literal 0	0	1	0
(2)	0	0	0	3
(3)	0	0	0	1
(4)	0	0	0	1
(2, 1)	0	0	0	2
(2, 2)	0	0	0	2
(2, 3)	I	0	0	0
(2, 1, 1)	C	1	0	0
(2, 1, 2)	the address of the literal D	0	1	0
(2, 2, 1)	E	0	0	0
(2, 2, 2)	F or zero	0	0	0
(3, 1)	J	0	0	0
(4, 1)	K	0	0	0

PARAMETER INDEX	CQ OR CP
(0)	= the address of the literal 1 or 0, ((@C, =the address of the literal D), (E, F*(G<H)), I),@(J), (K)
(1)	= the address of the literal 1 or 0, depending on whether the value of A equals the value of B
(2)	((@C, =the address of the literal D), (E, F*(G<H)), I)
(3)	@(J)
(4)	(K)
(2, 1)	(@C, =the address of the literal D)
(2, 2)	(E, F*(G<H))
(2, 3)	I
(2, 1, 1)	@C
(2, 1, 2)	= the address of the literal D
(2, 2, 1)	E
(2, 2, 2)	F or zero, depending on whether the value of G is less than the value of H
(3, 1)	J
(4, 1)	K

PARAMETER INDEX	RQ OR RP
(0)	the address of the literal 1 or 0, ((@C, the address of the literal D), (E, F*(G<H)), I), @(J), (K)
(1)	the address of the literal 1 or 0, depending on whether the value of A equals the value of B
(2)	(@C, the address of the literal D), (E, F*(G<H)), I
(3)	J
(4)	K
(2, 1)	@C, the address of the literal D
(2, 2)	E, F*(G<H)
(2, 3)	I
(2, 1, 1)	C
(2, 1, 2)	the address of the literal D
(2, 2, 1)	E
(2, 2, 2)	F or zero, depending on whether the value of G is less than the value of H
(3, 1)	J
(4, 1)	K

2-41. PROGRAM SECTIONS

An assembly may be divided into logical subdivisions called sections. Each section has a protection key for use in regrouping the various sections of an assembly at object time. A section is defined by the SEC directive; see Topic 4-15.

Sections provide the basis for addressing memory locations during an assembly. Memory locations are identified in the Assembler as relative locations from the start of the section.

Section numbers are assigned to each section by the Assembler. Section numbers 1 through 63 may be assigned in one assembly; i.e., any given assembly may have a maximum of 63 sections.

2-42. RELOCATION

Since the Assembler does not actually place object statements in fixed Central Memory locations, the relative locations assigned by the Assembler must be re-

locatable to available memory locations. Thus the relative location of a statement within a section is a relocatable value, and the value of a symbol, or an expression that refers to a relative location, is a section identification and the relative location within that section.

2-43. CONSTANTS

Two types of constants are identifiable during an assembly: (1) the actual value of a numeral, and (2) the relative location of a symbol within its section. The relative location of a symbol in its section is referred to as an address constant.

2-44. Address Constants

There are two classifications of address constants: (1) an address constant that is an internally relocatable value; i.e., a value whose section and relative location within its section is defined in the current assembly; and (2) an address constant that is an externally relocatable value; i.e., a value whose section and whose relative location within that section is defined in another assembly.

2-45. Numeral Constants

The value of a numeral is not relocatable. An absolute value cannot be defined as belonging to a section or to a relative location within a section. An absolute value may result from the use of relocatable items in an expression which produces loss of identity of the items within their sections (see Table 2-5).

2-46. LOCATION COUNTER

The location counter is a relocatable variable whose value is the current section number and current relative location within that section. The value of the location counter is positioned at the statement being assembled. The character, $, represents the value of the location counter symbolically. Use of $ in the operand of various control directives (see Topic 4-12.) permits the value of the location counter to be changed so that assembly control may be changed to different sections or to other positions within the same section.

2-47. RELOCATABILITY OF SYMBOLS

The section to which a symbol belongs is determined by either of the following: (1) the symbol may be equated to a procedure reference statement, or, (2) it may be equated to the value in the location counter. Such symbols are relocatable since the statement's location will be relocated with the section itself. Symbols defined in other assemblies and identified by use of the EXTRN directive (see Topic 4-5) are relocatable.

Symbols which are equated to absolute expressions or items are absolute. Symbols equated to the T(a) intrinsic function are absolute.

2-48. RELOCATABILITY OF EXPRESSIONS

Expressions, because they contain symbols, may be evaluated as absolute or relocatable. An expression that appears in more than one section because the symbols in the expression are defined in different sections is illegal, e.g., A+B where A and B are relocatable and belong to separate sections.

Table 2-5 shows, for each type of operator, the relocatability of the result. The result may be relocatable, absolute, or illegal. If the result is relocatable, its section is the section of the relocatable item or items.

2-49. Effect of Relational Operators

The effect of relational operations (i.e., initiated by the operators: <, ¬<, =, ¬=, <=, >, ¬>, >=) is as follows:

A	B	A REL B
ABS	ABS	will compare, if same length
ABS	RELOC	will not compare, evaluated as false
RELOC	ABS	will not compare, evaluated as false
RELOC	RELOC	will compare if in same section

Note: Since the result of a relational operation is always zero (false) or one (true), the result is always absolute.

Table 2-5. Results of Operations on Absolute and Relocatable Items in Expressions

A	B	A+B	A-B	A*B	A/B
ABS	ABS	ABS	ABS	ABS	ABS(B≠0)
ABS	RELOC	RELOC	illegal	Note1	illegal
RELOC	ABS	RELOC	RELOC	Note2	Note3
RELOC	RELOC	illegal	Note4	illegal	illegal

A	B	A++B	A--B	A**B	A==B
ABS	ABS	ABS	ABS	ABS	ABS
ABS	RELOC	illegal	illegal	illegal	illegal
RELOC	ABS	illegal	illegal	illegal	illegal
RELOC	RELOC	illegal	illegal	illegal	illegal

Note 1: Illegal unless A equals zero or one. If A is one, the result is relocatable; if A is zero, the result is an absolute zero.

Note 2: Illegal unless B equals zero or one. If B is one, the result is relocatable; if B is zero, the result is an absolute zero.

Note 3: Illegal unless B equals one. If B equals one, the result is relocatable.

Note 4: Illegal unless A and B are in the same section. If A and B are in the same section, the result is absolute.

SECTION III
LANGUAGE STRUCTURE

3-1. STATEMENT FORMAT

A program consists of a sequence of coded lines, each of which contains from 1 to 80 characters. However, only the first 72 characters of a line are processed by the Assembler. A line may contain a statement or a comment. Statement columns 73 through 80 can be used for program identification or sequence numbering.

A statement generally consists of three coding fields: a label field, a command field, and an operand field. These three fields are of variable length and are terminated by one or more blanks; i.e., no embedded blanks are permitted in these fields. Any statement columns to the right of the operand field may be used as a remark field which contains text. There is an optional code field for the PROC and NAME directives which does not apply to any other statements.

GENERAL FORM: The general form of an Assembler statement is:

Label	Command	Operands	Remarks
[symbol] ␣	symbol ␣	[exp1[, exp2[, . . . , expn] ␣	[text]

Examples:

TEST	SEC	0	
START	LDF	CINIT, X1L	
BEGIN	LF	#10, INIT, X1	

3-2. CONVENTIONS FOR DESCRIBING LANGUAGE STATEMENTS

The following conventions are used to illustrate the language statements:

1. Upper case letters and punctuation marks (except those explained in items 3 and 4 below) represent information that must be coded exactly as shown.

2. Lower case letters and words are generic terms that represent information that must be supplied; i.e., a substitution must be made when coding a parameter or option so represented.
3. Information within brackets [] is optional. It may be included or omitted entirely, depending upon program requirements.
4. When several choices are enclosed in braces{}, one of the enclosed alternatives must be selected by the programmer. If one of the alternatives is underlined, the parameter may be omitted and the system assumes the underlined alternative.
5. Mandatory blanks are represented by a slashed, lower-case letter "b" (/b). This symbol is not used to represent permissible blanks.

3-3. CONTINUATION LINES

A semicolon (;) appearing in the operand field is a line terminator which signals that the following line is to be treated as a continuation of the current one. That is, the semicolon is considered to be followed immediately by the first non-blank on the following line, and the information following a semicolon on the line on which the semicolon appeared is ignored.

Restrictions: No more than two continuation lines are permitted for each statement.

A semicolon (;) cannot be used to terminate an item; such usage will be treated as an error.

Character strings and intrinsic functions are the only types of items that can be divided by a semicolon. Character strings can be divided anywhere within the string, but leading blanks on the continuation line are not treated as part of the character string. Intrinsic functions can be divided by a semicolon only after the open parenthesis.

Examples: The following lines of code represent use of continuation lines:

```
CMPREG   L   (A1*(SUM1=SUM2))++(A2*(SUM1>SUM2))++(A3*(SUM1<SUM2)),(D(;
SUM1),B2),X5   COMPUTE REGISTER TO LOAD
MSSG   DATA   "THIS COMPUTATION EXTENDS INTO AN UN;
DEFINED REGION."
```

3-4. LABEL FIELD

A statement may be given a name by the programmer to permit references to be made to the statement from other points within the program. The use of name is normally optional, but some directives do require a symbol in the label field. The label must start in column 1. If no label is used, column 1 must be blank.

3-5. Reserved Symbols

The following symbols are reserved and may not be used as labels:

1. Symbolic register names; viz.:
   a. B0 through B15, A0 through A15, X0 through X7, and V0 through V7 for Central Processor assemblies, and
   b. R0 through R3, R0L through R3L, R0R through R3R, R0B0 through R3B3, C0 through C63, C0L through C63L, C0R through C63R, and C0B0 through C63B3 for Peripheral Processor assemblies.
2. The names of any of the directives, e.g., DATA, FORM, etc.
3. The names of any of the built-in procedures for the machine instructions; i.e., the Assembler mnemonics for the machine operation.
4. A name assigned to a procedure within an assembly becomes a reserved symbol for that assembly.

WARNING: Because the Simulator's reserved symbol table contains both Peripheral Processor and Central Processor reserved symbols (such as register labels), it is advisable to avoid use of these symbols as labels in any assemblies.

3-6. Intrinsic Functions as Labels

An intrinsic function may be used as the label of a statement within a procedure to refer to an operand of the procedure definition or procedure calling statement.

During assembly the operand is then substituted for the intrinsic function. The operand in this case cannot be a list.

Example:

LABEL	COMMAND	OPERANDS
PROG	PROC	OP1, OP2, OP3
	. . .	
CP(1)	LD	R1, START
	. . .	
	PEND	PROG
	. . .	
	PROG	SUM1, SUM2, SUM3
	. . .	

results in:

	. . .	
SUM1	LD	R1, START
	. . .	

3-7. COMMAND FIELD

Each statement has a command. The command begins with the first non-blank following the label field and is terminated by one or more blanks. The command must be represented symbolically.

The command field dictates the operation to be performed and may call an assembler directive or a previously defined or built-in procedure. Thus the command field contains a mnemonic which is the name of a directive or a label of a procedure. New operations may be introduced by defining new procedures. (The ASC Peripheral Processor and Central Processor instruction sets are represented by built-in procedures.)

Any error occurring in the command field will result in an illegal instruction. The Assembler will generate one word of zeros (absolute) of loader text and will process the operand field for general syntax errors only.

3-8. Intrinsic Functions as Commands

An intrinsic function may be used as the command in a statement within a procedure to refer to an operand of the PROC statement, the NAME statement, or the statement that calls the procedure. That operand in such a case would be a command itself (for example, a load instruction) which, during the expansion of the procedure, would be substituted in place of the intrinsic function that called it. An intrinsic function used in the command field cannot be replaced with a list.

Example:

LABEL	COMMAND	OPERANDS
PROG	PROC	LD, ST, STOP, START
	. . .	
	CQ(1)	R1, VQ(4)
	. . .	
	PEND	PROG
	. . .	
	PROG	
	. . .	

results in:

	. . .	
	LD	R1, START
	. . .	

3-9. OPERAND FIELD

Most commands require operands. If a line is to include operands, the operand field begins with the first non-blank following the command field.

The operand field is composed of a list of elements. Elements are composed of one or more expressions (often referred to as parameters). The last element in the operand is terminated by a blank; all other elements in lists are terminated by a comma. Sublists (elements in the form of lists enclosed in parentheses) may exist. Intrinsic functions may also be elements in an operand.

Examples:

OPERAND	COMMENT
250	This operand is a list which is the trivial case of one item.
A+2, 2	This operand contains a list of two elements or expressions.
A1, (B, C)	This operand contains a list of two elements, the second of which is a sublist of two expressions.
RP(2, 1), 3	This operand contains a list of two elements, the first of which is an intrinsic function.

Elements omitted from the right end of an operand list are assumed to have a value of zero, (0, 0, 0 may be written as 0). If the operand field is left vacant, the remark field must also be left vacant (blank). The number of blanks between fields is not limited.

The Assembler will check each expression in the operand field for valid syntax. If a syntax error is found, the Assembler will print a diagnostic flag and supply zero in place of the expression found to be in error. The object code generated for the remainder of the statement depends upon the use of the expression; i.e., the command produced may or may not be correct. In some cases, a word of zeros is generated.

Note: On the PROC and the NAME card images, columns 71 and 72 of the first card compose a special field. Therefore, if the operand needs to extend beyond column 70 on these statements, insertion of a semicolon in or before column 70 of the first card allows the use of continuation cards.

3-10. REMARK FIELD

The optional remark field is allowed as a convenience for documentation and has no effect upon the nature of the assembled object code. A remark must be isolated from the end of the operand field by at least one blank. The remark field may not be continued onto the next line.

Note: On the PROC or the NAME card images, columns 71 and 72 of the first card image compose a special field. Therefore, the remark field cannot extend beyond column 70 of the first card of one of these statements. Use of the remark field is normal if begun on continuation cards.

3-11. CODE FIELD

The optional code field is a special field for the PROC and NAME directives that indicate to the Assembler the number of words of code that will be generated when the procedure is expanded. This field comprises columns 71 and 72 of the first card image in a PROC or a NAME directive. (Topics 6-2 and 6-3)

3-12. COMMENT LINES

A line (which is not a statement continuation line) whose first column contains an asterisk (*) is treated as entirely commentary. No loader text is generated. The line will be listed in context.

3-13. BLANK LINES

A line consisting of only spaces (blanks) in character positions 1 through 71 is treated as commentary. A blank line will be printed on the assembly listing as a result.

SECTION IV
DIRECTIVES

4-1. INTRODUCTION

Assembler directives supply special types of information to the Assembler. A reference to any symbolic item in an expression on a directive line must have previously appeared as a label; e.g., it must be possible to evaluate immediately the expression(s) in the operand field of the directive. If the operand expression can not be evaluated, it will be assigned a value of zero, and an error message will be printed.

Exceptions to this rule are the symbols in the operands of the EXTRN, the ENTRY, the GOTO, the DATA, and the USING directives. These symbols may be forward references. The SET directive has conditional exceptions. All forward references must be satisfied with values which do not exceed one word. The value of a forward reference may be relocatable or absolute, and, if n is the value, $-2^{31} \leq n < 2^{31}$.

Note: In the general forms of the directive statements, items enclosed in brackets are optional.

4-2. DEFINITION DIRECTIVES

4-3. EQUATE DIRECTIVE (EQU)

The EQU directive is used to assign a permanent value to its symbolic label.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
symbol	⊬	EQU	⊬	exp

The symbolic label is defined to have the value of the expression, exp.

The value of exp may be absolute or relocatable, positive or negative, and is expanded to an integral multiple of fullwords.

Restrictions: An EQU statement must have a label.

Symbols in the expression must be defined prior to their use in the EQU directive; i.e., the expression cannot be a forward reference.

Symbols defined in the label of this directive cannot be redefined.

Limitations: The maximum number of characters in a character string named by an EQU statement is 28.

Examples:

LABEL	COMMAND	OPERANDS
ALPHA	EQU	$+3
BETA	EQU	ALPHA
NO	EQU	0

4-4. SET DIRECTIVE (SET)

The SET directive is used to assign a temporary value to its symbolic label.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
symbol	⍽	SET	⍽	exp

The symbolic label represents the value of the expression, exp.

The value of the symbol can be changed by redefining it as the label of another SET directive.

The value of exp may be absolute or relocatable, positive or negative, and is expanded to an integral multiple of fullwords. The expression may contain a symbol that is a forward reference.

Restrictions: A SET statement must have a label.

If the value of the symbolic label is changed by being used as the label of another SET directive, no subsequent values can exceed the length of the first.

Limitations: If exp contains a forward reference, n, the value of n is within the range: $-2^{31} \leq n < 2^{31} - 1$.

The maximum number of characters in a character string named by a SET statement is 28.

Examples:

LABEL	␣	COMMAND	␣	OPERANDS
ALPHA		SET		3
YES		SET		BETA-ALPHA
BETA		SET		$

4-5. EXTERNAL NAME DIRECTIVE (EXTRN)

The EXTRN directive is used to identify every symbol that is used but not defined in the current assembly.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	␣	EXTRN	␣	symbol[,...[,symbol]]

Each name appearing in the operand field of the EXTRN directive will be output to the Link Editor, provided that reference to that name is within the current assembly. Any declared external names to which references are not made will not be output to the Link Editor. An external name must be referenced by an address constant.

WARNING: Any reference within the current assembly to an external name not declared by an EXTRN directive will be treated as being undefined, and an error flag will appear on the assembly listing.

Restrictions: An EXTRN statement cannot have a label.

Limitations: A limit of 255 external symbols may be declared for a module.

A limit of 15 external symbols may be listed in the operand field of one EXTRN statement.

Examples:

LABEL	∅	COMMAND	∅	OPERANDS
		EXTRN		ALPHA
		EXTRN		SQRT, SIN

4-6. ENTRY NAME DIRECTIVE (ENTRY)

The ENTRY directive is used to establish linkages between programs that have been assembled separately but are to be loaded and executed together.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	∅	ENTRY	∅	symbol[,...[symbol]]

Each name appearing in the operand field of the ENTRY directive declares an entry point into the current assembly to which external programs may refer. Any name declared to be an entry point that is not defined within the assembly will cause an error message to be output in the assembly listing.

Control section names can be used as entry points. Entry points are generated automatically for them.

Restrictions: An ENTRY statement cannot have a label.

Each name appearing in the operand field of the ENTRY directive must also appear as the label of a statement in the body of the assembly and must have a relocatable value defined in a control section.

Limitations: A limit of 255 entry names may be declared for a module.

A limit of 15 entry names may be listed in the operand field of one ENTRY statement.

Examples:

LABEL	COMMAND	OPERANDS
	ENTRY	ALPHA
	ENTRY	SQRT, SIN

4-7. DATA DIRECTIVE (DATA)

The DATA directive generates enough fullword data units to contain the information in the operand field.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
[symbol]	DATA	exp[, . . . [, exp]]

The label symbol is the location of the first expression in the operand list. The symbol is given the current value of the location counter.

Each expression is expanded to a multiple of fullword units.

An address constant is generated for any expression that is not absolute.

If a generated address constant refers to an external symbol, the output module indicates that the value of the external is to be added, at link edit (or simulation) time, to the constant displacement derived from the expression in which the external symbol is used; e.g., for the statement

```
DATA    EXTERN1+10,EXTRN2+(T<S)
```

in which EXTRN1 and EXTRN2 are the first and second external symbols defined in the assembly, ten will be added to the location of EXTRN1 at link edit time, and either zero or one will be added to the location of EXTRN2, depending upon whether (T < S) is false or true, respectively.

Limitations: A limit of 15 expressions may be listed in the operand field of one DATA statement.

A symbol in the operand field of the data statement that is defined by an EQU or SET statement is assumed to have a singleword value. If a symbol is set (EQU or SET)

to a value (e.g., a character string) greater than one word in length, only the rightmost word of the value will be generated as data.

Examples:

LABEL	⍽	COMMAND	⍽	OPERANDS
		DATA		BETA
ALPHA		DATA		0
FLTPNT		DATA		1.0,2.0
		DATA		"LITERAL"
		DATA		1.0D0

4-8. FORMAT DIRECTIVE (FORM)

The FORM directive is used to specify arbitrary data formats.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
symbol	⍽	FORM	⍽	exp1, exp2[, exp3, ..., expn]

The values of the absolute expressions in the operand field of the FORM directive give the bit lengths of successive fields in the resultant data word.

Reference may be made to a format definition by using its label as the command in any succeeding statement with an operand field composed of values to be placed in the object word fields defined in the FORM statement.

Restrictions: A FORM statement must have a label.

The sum of the values of the expressions must be a multiple of fullword (32 bit) units.

Limitations: A limit of 15 fields may be defined in the operand list of a FORM statement.

Examples:

LABEL	↓	COMMAND	↓	OPERANDS
FSTART		FORM		8, 4, 4, 4, 12
		FSTART		#C4, 2, 8, #E, 31

produces the hexadecimal code:

BYTE	0		1		2		3	
	C	4	2	8	E	0	1	F
HEX	0	1	2	3	4	5	6	7

4-9. USING DIRECTIVE (USING)

The USING directive indicates to the Assembler that the specified base register contains the value of the relocatable expression.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	USING	↓	exp, register

The Assembler will select a base and compute a displacement from the specified base value for each relocatable expression that follows the USING statement. The base selected will be that base (for the section) which produces the smallest displacement.

Failure to specify a base register or registers for each section of an assembly will result in addressability errors.

Restrictions: A USING statement cannot have a label.

A base register with a symbolic value of zero cannot be specified as the register operand of the USING directive.

Note: If the defined machine (viz., the Peripheral Processor) has no base registers, use of the USING directive is illegal.

Examples:

LABEL	COMMAND	OPERANDS
	USING	ALPHA, B2
	USING	$+1, B14

4-10. DROP DIRECTIVE (DROP)

The DROP directive indicates to the Assembler that the specified base register is no longer available for base selection. The base will not be considered available until another USING directive declares it to be available.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
blank	DROP	register

Restrictions: A DROP statement cannot have a label.

Note: If the defined machine (viz., the Peripheral Processor) has no base register, use of the DROP directive is invalid.

Examples:

LABEL	COMMAND	OPERANDS
	DROP	B14

4-11. ORIGIN DIRECTIVE (ORG)

The ORG directive is used to set or reset the location of the origin for all or a portion of the section being assembled.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
[symbol]	ORG	exp

The location counter is set to the value of the expression, exp. All code generated following the ORG directive will begin at the location whose value is that of the expression.

If the expression is blank, it directs the Assembler to use the highest value previously assigned to the location counter of the section being assembled as the present value of the location counter.

The label, if present, is assigned the value of the location counter before the location counter is reset.

Restrictions: The expression must have a relocatable value and must be within the same control section as the ORG statement.

Examples:

LABEL	⌀	COMMAND	⌀	OPERANDS
ALPHA		ORG		$+50
		ORG		ALPHA

4-12. CONTROL DIRECTIVES

4-13. LITERAL ORIGIN DIRECTIVE (LITORG)

The LITORG directive sets the location of the origin for all literals (regardless of the referring section) defined since the previous LITORG directive and places the locations of the literals in their respective object code statements.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
[symbol]	⌀	LITORG	⌀	[exp]

The origin is determined by incrementing the value of the expression, if necessary, to a doubleword boundary. The literals will be generated beginning at the aligned location.

The label, if present, is assigned the value of the location counter before the location counter is reset.

After the literals have been generated, the location counter will remain set to the first location following the last generated literal.

If the expression is blank, the current location counter value becomes the literal origin before alignment.

Restrictions: The expression must have a relocatable value and must be within the same control section as the LITORG statement.

Examples:

LABEL	⊬	COMMAND	⊬	OPERANDS
ALPHA		LITORG		$+10
		LITORG		

4-14. END ASSEMBLY DIRECTIVE (END)

The END directive signals termination of the assembly.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	⊬	END	⊬	[exp]

The value of the expression represents the beginning execution address of the assembly when it is loaded and run (unless otherwise overridden). If the operand field is blank, no address for beginning execution of the program is output to the loader.

Whenever an END statement is encountered, it will be recognized as the end of the assembly.

Restrictions: An END statement cannot have a label.

The END directive cannot be used in a procedure.

The expression, exp, must be relocatable.

Examples:

LABEL	⊬	COMMAND	⊬	OPERANDS
		END		FIRST
		END		

4-15. SECTION DIRECTIVE (SEC)

The SEC directive defines a control section and asserts assembly control to that section for the generation of any subsequent code that is to have the same protection conditions.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
[symbol]	↓	SEC	↓	[exp1][,exp2]

The label is the symbolic name applied to the control section. Subsequent uses of a SEC statement with the same label will return control to the section at the location immediately following the highest location count used previously within the section. Sections may be resumed as desired.

The Assembler assigns a control section number to the section when the defining statement is used for the first time. Section numbers are assigned sequentially.

Initially, the location counter is set to zero. An ORG directive (see Topic 4-11) may be used to adjust the location counter values.

The first operand expression, exp1, specifies the hardware protection of all code generated under control of the defined section. Expression 1 need not be used in subsequent returns to a defined section; the original protection will be assumed. Expression 1 must be absolute with a value of 0, 1, 2, or 3. An exp1 value of 0 specifies read, write, or execute (i.e., no protection); an exp1 value of 1 specifies read only; an exp1 value of 2 specifies read or write; and an exp1 value of 3 specifies execute only. If exp1 is blank, the value 0 is assumed.

The second operand expression, exp2, specifies the memory alignment for the beginning of the section. Expression2 must be absolute and the value is considered to be an exponent of 2, i.e., 2^{exp2}. If exp2 is not present, a value of 3 is assumed, and, thus, the section will be aligned on an octet boundary.

The Assembler will assign a control section to the module name if no SEC directive is used. No protection (i.e., read, write, or execute) will be assumed for such a section.

Restrictions: Both expressions must be absolute.

The protection condition cannot be changed after the initial defining SEC statement; i.e., a given section has only one protection condition.

Limitations: Only one unlabeled SEC statement is allowed.

WARNING: The Assembler will create absolute literals in a section with execute only protection even though they cannot be read and, therefore, cannot be used.

Examples:

LABEL	COMMAND	OPERANDS
ALPHA	SEC	1,2
BETA	SEC	0
	SEC	3,3

4-16. COMMON MODULE DIRECTIVE (COM)

The COM directive defines a common module.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
[symbol]	COM	[exp1][,exp2]

The label is the symbolic name applied to the common section. Subsequent uses of a COM statement with the same label will return control to the common section at the location immediately following the highest location count used previously within the section. Common sections may be resumed as desired. If no label is present, "blank" common is defined; i.e., it is an unlabeled common section.

The protection condition of the common section is specified by exp1. The absolute value of exp1 may be 0, 1, 2, or 3 with the same protection interpretation as for the SEC directive, viz., no protection, read only, read or write only, and execute only, respectively. If exp1 is blank, zero is assumed.

The beginning boundary alignment is specified by exp2 where exp2 is an exponent of 2; i.e., the alignment will be 2^{exp2}. If exp2 is blank, three (an octet boundary) is assumed.

The module definition output generated for a COM statement has the same format as that for a SEC statement; the setting of a reserved bit within the format distinguishes the defined module as a common module.

Restrictions: Both expressions must be absolute.

The protection condition cannot be changed after the initial defining COM statement.

Examples:

LABEL	COMMAND	OPERANDS
ALPHA	COM	0,3
BETA	COM	
	COM	1

4-17. DUMMY SECTION DIRECTIVE (DUM)

The DUM directive defines an absolute dummy section.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
[symbol]	DUM	[exp1][,exp2]

Any reference to the absolute dummy section name or to any symbol defined within the dummy section is treated as an absolute reference to a section. The symbolic name has a value of zero since it is the first location in the dummy section. The values of any other symbols defined (as labels of statements) within the section have the values of their respective displacements from the beginning of the dummy section.

A dummy section produces no object text output, and no evidence will exist in the object "deck" that the DUM statement appeared in the source file.

The operand expressions have no significance other than as a comment for the protection and boundary conditions of the actual section for which the dummy section substitutes.

Examples:

LABEL	COMMAND	OPERANDS
ALPHA	DUM	2,3
BETA	DUM	0,0
	DUM	

4-18. DUMMY COMMON MODULE DIRECTIVE (COMD)

The COMD directive defines a relocatable dummy section.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
symbol	COMD	[exp1][, exp2]

Any reference to the dummy common module name or to any symbol defined within the dummy common section is treated as a relocatable value to which the value of the symbolic label is to be added at link-edit time. At assembly time the symbolic label has the value of relative zero, and symbols defined (as labels of statements) within the dummy common section have values that are the relocatable displacements relative to the beginning of the module.

The symbolic label is assumed to be the name of a common section.

A dummy common module produces no object text output.

The operand expressions have no significance other than as a comment for the protection and boundary conditions of the actual common section for which the dummy common section substitutes.

Restrictions: A COMD statement must have a label.

Examples:

LABEL	↓	COMMAND	↓	OPERANDS
ALPHA		COMD		0, 3
BETA		COMD		1, 1

4-19. COPY DIRECTIVE (COPY)

The COPY directive causes the specified file, "sourcefilename," to be copied inline as source text to the Assembler.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	COPY	↓	sourcefilename

The source statements from the file, sourcefilename, are merged into the assembly after the COPY statement and before any later source statements in the assembly. The file may exist on an indicated user library or on the system procedure library.

The COPY function is processed during PASS 1 of the Assembler without regard to level of assembly.

A COPY statement within a procedure definition will cause the included text to become part of the procedure definition.

Restrictions: A COPY statement cannot have a label.

The occurrence of an END statement in the copied file will cause termination of the assembly.

Limitations: The COPY function cannot be used to copy part of a source file; all card images in the file will be copied.

Examples:

LABEL	↓	COMMAND	↓	OPERANDS
		COPY		SOURCE
		COPY		PROC1

4-20. RESERVE DIRECTIVE (RES)

The RES directive is used to reserve space within the assembly.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
[symbol]	↓	RES	↓	exp

The present value of the current location counter is modified by the value of the expression, exp. The expression may be positive or negative, but must be absolute.

The value of the symbolic label is the value of the current location counter before it is modified.

Restrictions: The expression, exp, must be absolute.

Limitations: The maximum value of exp is 65536.

Examples:

LABEL	↓	COMMAND	↓	OPERANDS
ALPHA		RES		10
BETA		RES		#16
		RES		100

4-21. ALIGN DIRECTIVE (ALIGN)

The ALIGN directive causes the location counter value to be incremented, if necessary, to place the next statement on a specified word boundary.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	ALIGN	↓	exp1, exp2

The second operand expression, exp2, specifies a basic boundary alignment, and the first operand specifies a number of words past that basic alignment; e.g., ALIGN 2, 8 would specify the second word past an octet boundary.

If the current value of the location counter is not on a word boundary as specified by exp2 and expl, the location counter value will be incremented by the least value sufficient to place the new location count on the specified boundary. The contents of any locations skipped are not modified.

Mathematically, the location count must meet the criterion: $C \equiv x \bmod (y)$, where C is the location count, x is the value of expl, and y is the value of exp2. Illustratively, this means that ALIGN 2, 8 will force the location counter value to a member of the set $\{2, 10, 18, 26, \cdots\}$.

Restrictions: An ALIGN statement cannot have a label.

The expressions, expl and exp2, must be absolute.

Expl must be less than exp2.

Examples:

LABEL	COMMAND	OPERANDS
	ALIGN	0, 8
	ALIGN	1, 2
	ALIGN	0, 16

4-22. DO DIRECTIVE (DO)

The DO directive provides control of assembly by including, excluding, or repeating a variable number of statements. The result in the assembly is the same as if the "DO-controlled" statements had been included, excluded, or repeated in the source input stream.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
[symbol]	DO	[expl][,[exp2][,[exp3]]]

ACTION SUMMARY: Letting the DO parameters be represented by x, y, and z, respectively, the action of the DO directive when used in procedures may be summarized as:

1. If x>0, assemble the next y statements x times and skip z statements following the last statement assembled.

2. If x≤0, skip the next y statements; ignore z.

When used outside of procedures, the DO directive is more limited in its interpretation and its action may be summarized as:

1. If x>1, assemble the next statement x times; ignore y and z.

2. If x = 1, assemble the next y statements once and skip z statements.

3. If x< 1, skip the next y statements; ignore z.

Restrictions: No statements within the range of a given DO statement other than a nested DO statement, an SNAME statement, or a SET statement may have a label.

Limitations: When the DO directive is used outside of procedures:

1. If exp1 is greather than one, exp2 will be processed as having a value of one and exp3 will be processed as having a value of zero regardless of how they are coded.

2. Of the intrinsic functions, only D(exp) and T(exp) are valid as parameters of a DO statement in a Peripheral Processor assembly, and only B(exp), D(exp), and T(exp) are valid in a Central Processor assembly.

3. A DO directive may be nested only to permit exclusion of the nested DO statement; a nested DO statement's control range will not be repeated.

4. The SNAME and GOTO directives (see Topics 6-5 and 6-6) are not valid.

Default Values for Parameters: The following table illustrates the Assembler's interpretation of the DO parameters when the DO directive is used in a procedure:

	expl ≤ 0	expl > 0
CODED	ASSUMES	ASSUMES
DO x	DO 0, 1, 0	DO x, 1, 0
DO x,	DO 0, 1, 0	DO x, 1, 0
DO x, ,	DO 0, 0, 0	DO x, 0, 0
DO x, , z	DO 0, 0, 0	DO x, 0, z
DO x, y	DO 0, y, 0	DO x, y, 0
DO x, y,	DO 0, y, 0	DO x, y, 0
DO x, y, z	DO 0, y, 0	DO x, y, z

The following table illustrates the Assembler's interpretation of the DO parameters when the DO directive is used outside a procedure:

	expl < 1	expl = 1	expl > 1
CODED	ASSUMES	ASSUMES	ASSUMES
DO x	DO x, 1, 0	DO 1, 1, 0	DO x, 1, 0
DO x,	DO x, 1, 0	DO 1, 1, 0	DO x, 1, 0
DO x, ,	DO x, 0, 0	DO 1, 0, 0	DO x, 1, 0
DO x, , z	DO x, 0, 0	DO 1, 0, z	DO x, 1, 0
DO x, y	DO x, y, 0	DO 1, y, 0	DO x, 1, 0
DO x, y,	DO x, y, 0	DO 1, y, 0	DO x, 1, 0
DO x, y, z	DO x, y, 0	DO 1, y, z	DO x, 1, 0

Note: If expl is defaulted, it will be assumed to be zero.

ITERATION GROUP: The value of exp2 specifies the number of statements to be assembled as a group. This group, called the iteration group, begins with the statement immediately following the DO statement.

Restrictions: Outside of procedures, if expl is greater than one, exp2 defaults to one.

Limitations: The value of exp2 is limited to the range: $0 \leq exp2 \leq 255$.

Note: Commentary lines are not statements and are, therefore, ignored in DO directive iterations.

ITERATION COUNT: The value of exp1 specifies the number of times the iteration group is to be assembled. This value is called the iteration count.

An exp1 value of less than one causes the iteration group to be skipped without action. An exp1 value of n, where n is greater than or equal to one, causes the iteration group to be assembled n consecutive times.

Restrictions: Outside of procedures, if exp1 is greater than one, exp2 defaults to one; if exp1 is equal to or less than one, exp2 may be greater than one.

Limitations: The effective value of exp1 is limited to the range: $0 \leq exp1 \leq 255$. Its actual value is limited to the range: $-255 \leq exp1 \leq 255$.

SKIP COUNT: The value of exp3 specifies the number of contiguous statements in the source stream that are to be excluded when the iterations are complete. This value is called the skip count.

The statements skipped are those that immediately follow the last statement iterated.

If exp1 is less than one, the DO execution is complete after the iteration group is skipped and exp3 is ignored; i.e., only the number of statements equal to exp2 will be skipped.

Restrictions: Outside of procedures, exp3 is defaulted to zero for all cases other than those in which exp1 is equal to one.

Limitations: The value of exp3 is limited to the range: $0 \leq exp3 \leq 255$.

SATISFACTION OF PARAMETERS: When a number of statements equal to (or greater than) the value of exp2 have been assembled in one iteration, exp2 is said to be satisfied for that iteration.

When a number of iterations equal to the value of exp1 have been completed, exp1 is said to be satisfied for that DO statement.

The DO directive is said to be satisfied when the iterations and/or skips specified by all three parameters have been completed.

DO LABEL: A label on the DO directive provides symbolic access to the number of the iteration the DO directive is performing (or has performed) at the time reference is made to the label.

When the DO statement has been encountered in the source stream, the label is initially given a value of one; thus, if the DO label is used as one of the parameters of its own DO statement, that parameter will always be evaluated as one. This initial assignment of the label value overrides any previous assignment of a value to that symbol by a previous SET or DO statement.

The value of the DO label is incremented at the beginning of each iteration of the DO directive; thus, the value of the label is always the number of the iteration being performed, or is the number of the last iteration performed once the DO directive is satisfied. Any attempts to modify the value of the DO label by a SET statement or as the label of a nested DO statement within the range of the DO directive will cause an anomalous assembly. In summary, if a DO statement has an iteration count (value of expl) of n and has a label, the value of the label will be incremented through the series $\{1, 2, 3, \ldots, n\}$ in successive iterations. If the value of expl is less than one, the label will always have the value of one.

Once the DO directive is satisfied, the label retains its last value unless or until its value is modified by a SET statement or it is used as the label of another DO statement, or that DO statement is reaccessed. Such modifications must occur outside the range of the subject DO directive.

Restrictions: The value of a DO label cannot be preset to a value other than one.

The value of a DO label cannot be modified within the range of the subject DO directive, other than by its own iteration incrementation.

Limitation: The range of values the DO label may acquire is: $1 \leq \text{label} \leq 255$.

NESTED DO DIRECTIVES: A DO directive is said to be nested when it is one of the statements included in the iteration group of another DO directive which is called the parent DO directive. DO directives may be nested up to 32 levels.

When a nested DO directive is encountered, it takes control from the parent DO directive and executes until it is satisfied, or until another nested DO directive is encountered. When the innermost nested DO directive is satisfied, it passes control to its immediate parent. The process of completion of a nested DO directive and the passing of control to its parent continues until the outermost DO directive has been satisfied and main level assembly resumes.

The statements in the iteration group of a nested DO directive, the statements the nested DO directive may skip, and the nested DO directive itself are all included in the count of statements used to determine satisfaction of exp2 of the parent DO directive. The iteration of the statements in the nested DO directive do not count as statements in the parent's iteration group. The nested DO directive retains control of assembly until it is satisfied before it passes control to its parent DO directive, even though, in the process of satisfying itself, it satisfies or more than satisfies the iteration group (exp2) of the parent DO directive.

Restrictions: An error message will be returned if the range of a nested DO directive exceeds the range of its parent DO directive, even though the resultant assembly may be the desired result.

Outside of procedures, a DO directive may be nested only to permit its exclusion from the assembly. The nested DO directive will not repeat statements within its range; neither will the parent DO directive cause the nested DO directive to repeat its action.

4-23. PSEUDO DIRECTIVES

4-24. INDIRECT ADDRESS CONSTANT DIRECTIVE (IND)

The IND directive is used to generate an indirect address constant.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
[symbol]	IND	exp1[, exp2[, exp3]]

Expression1 represents the address value and expression2 represents the second level index. Expression1 may be preceded by an at sign (@) to indicate another level of indirectness. If expression3 is present and the indirect is used by a branch instruction, the value of expression3 determines whether the branch is made to Central Memory or Read Only Memory. If the value of expression3 is zero, the branch is made to Central Memory. If the value of expression3 is non-zero, the branch is made to Read Only Memory. If expression3 is not present, a value of zero is assumed.

Note: Expression3 is not valid for the Central Processor.

Examples:

LABEL	COMMAND	OPERANDS
ALPHA	IND	BETA, XIR, 1
GAMMA	IND	@BETA
	IND	SIGMA

4-25. DATA HALFWORD DIRECTIVE (DATAH)

The DATAH directive will place the two values of expression1 and expression2 into the left and right halves, respectively, of the word generated by the statement. Both expressions must be absolute.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
[symbol]	DATAH	exp1, exp2

Examples:

LABEL	COMMAND	OPERANDS
ALPHA	DATAH	0, 1
	DATAH	-3, 2

4-26. LISTING DIRECTIVES

4-27. SKIP DIRECTIVE (SKIP)

The SKIP directive permits control of the assembly listing. This directive causes the Assembler to skip print lines or to eject the page of the assembly listing. The contents of the operand field also control the printing of the heading at the top of the new page. The directive itself is not printed on the listing.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	⍽	SKIP	⍽	[exp[, character string]]

If the expression is zero or blank, the page will be ejected. Otherwise, a number of print lines equal to the value of the expression will be skipped. The expression must be absolute.

SKIP directives are ignored if the NOLIST directive is used.

The operand field of the SKIP directive may have any of the following formats:

FORMAT	FUNCTION
SKIP	Eject the page; new page number equals old page number plus one; print previous title.
SKIP N	Skip N lines; if this causes page ejection, start at top of page as in previous format.
SKIP 0, "TITLE"	Eject the old page; new page number equals old page number plus one; new title is the character string, TITLE.
SKIP N, "TITLE"	Skip N lines; if this causes page ejection, start at top of new page; new title for the next page is the character string, TITLE, regardless of whether page is ejected now; new page number equals old page number plus one.

Restrictions: No label is allowed with the SKIP directive.

The expression must be absolute.

Limitations: The character string may not exceed 100 positions.

Examples:

LABEL	COMMAND	OPERANDS
	SKIP	3
	SKIP	
	SKIP	0,"TITLE"
	SKIP	6,"TITLE"

4-28. LIST DIRECTIVE (LIST)

The LIST directive is used to cause the object code listing to resume.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
blank	LIST	blank

The combination of NOLIST and LIST directives can be used when only a portion of the assembly listing is desired. The directive is not printed on the listing.

Restrictions: No label is allowed with the LIST directive.

Example:

LABEL	COMMAND	OPERANDS
	LIST	

4-29. NOLIST DIRECTIVE (NOLIST)

The NOLIST directive is used to suppress the listing.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	NOLIST	↓	blank

When the assembler encounters this directive, it stops the listing. The directive is not printed on the listing.

Restrictions: No label is allowed with the NOLIST directive.

Example:

LABEL	↓	COMMAND	↓	OPERANDS
		NOLIST		

SECTION V
ASSEMBLER OUTPUT

5-1. ASSEMBLER OUTPUT

The Assembler output consists of object modules and a listing with messages.

5-2. SOURCE PROGRAM LISTING

An assembly listing will be output in the format shown in Figure 5-1. A double space listing may be requested by control card option, or listing may be suppressed by control card option.

Columns with headings are provided for error flags, the location of each statement, the object text generated, the statement number, and each source statement. If program sequencing has been requested, an unheaded column of sequence numbers will be printed.

Error Column (ERRORS): The ERROR column contains the flags for Assembler error messages and procedure processing error messages. These error flags and their meanings are described in Topic 5-3.

Statement Location Column (LOCATION): The LOCATION column contains the location (in hexadecimal numerals) relative to the beginning of its section for each source statement.

Object Code Column (OBJECT TEXT GENERATED): The OBJECT TEXT GENERATED column contains the hexadecimal object code generated by each statement in the source program.

Statement Number Column (STMT): The STMT column contains decimal statement numbers for each source statement in an assembly.

```
T E X A S  I N S T R U M E N T S  A L A M O  --  C P U  A S S E M B L E R  L I S T I N G                    PAGE    1

ERRORS LOCATION OBJECT TEXT GENERATED          STMT              SOURCE STATEMENT                        02/05/70


                                               669
                                               670 MOVE      PROC
                                               671           L       A1,RP(1)
                                               672           ST      A1,RP(2)
                                               673           PEND    MOVE
                                               674           USING   TST,B1
                                               675 TST       SEC     0
      000000  14 1 0 0 1 008                   676           MOVE    A,B
      000001  24 1 0 0 1 009
      000002  14 1 0 1 1 00A                   677           MOVE    (C,X1),E
      000003  24 1 0 0 1 00B
                                               678 D         SET     D(ALPHA)
                                               679 G         SET     B(ALPHA)
                                               680 H         SET     D(BETA)
                                               681 J         SET     B(BETA)
      000004  14 1 0 0 1 00E                   682           MOVE    ((D,G)),F
      000005  24 1 0 0 1 00C
      000006  14 1 1 2 1 00E                   683           MOVE    (D(H,J),X2),K
      000007  24 1 0 0 1 00D
      000008                                   684 A         RES     1
      000009                                   685 B         RES     1
      00000A                                   686 C         RES     1
      00000B                                   687 E         RES     1
      00000C                                   688 F         RES     1
      00000D                                   689 K         RES     1
      00000E                                   690 ALPHA     RES     1
      00000F                                   691 BETA      RES     1
      000010  00000000                         692           END     TST


                                                     E N T R Y   N A M E S

                                                     TST       01 000000


ASSEMBLY COMPLETE.    NO STATEMENTS HAVE ERRORS.
```

Figure 5-1. Sample Source Program Listing

Note: The user should note that the Central Processor or Peripheral Processor Procedure Library statements which precede each program are not listed; therefore, statement numbers begin at some number greater than zero.

Source Statement Column (SOURCE STATEMENT): The SOURCE STATEMENT column contains each symbolic source statement in the assembly input.

SUMMARY OF SPECIAL LISTS: Following the listing of source statement for an assembly is a summary of special lists and the number of statements in error for each section. The format is as follows:

LITERALS ASSIGNED TO SECTION 01

location counter value literal

ENTRY NAMES

entry name location counter value

EXTERNAL NAMES

external name

ASSEMBLY COMPLETE xxxx STATEMENTS IN ERROR

ASSEMBLER AND PROCEDURE GENERATED MESSAGES

FLAG	ERROR CONDITION
D	DUPLICATE LABEL ASSIGNMENT
H	INVALID USE OF A DIRECTIVE
I	UNDEFINED INSTRUCTION
L	ERROR IN THE LABEL FIELD
U	UNDEFINED SYMBOL
W	WARNING, POSSIBLE ERROR
1	INVALID USE OF A REGISTER, OR INSUFFICIENT NUMBER OF PARAMETERS

The flags and error conditions are discussed in the following topic (5-3).

5-3. MESSAGES

An assembly listing line consists of the hexadecimal representation of the location counter and machine language instruction followed by the number and image of the original source statement. Message flags are indicated to the left of the location counter value.

A message flag is listed for each line in which an error condition is discovered. A word containing an error count is incremented by one. At the end of the assembly run, this count is set in the specified word on the listing. A maximum of six flags will appear on the listing for each line of generated listing.

5-4. Procedure Processing Generated Messages

The message flags following are generated by the Central Processor and the Peripheral Processor procedures that the Assembler processes. These flags are not generated by the Assembler itself.

5-5. CROSS-REFERENCE LISTING

A cross-reference listing will be output in the format appearing in Figure 5-2. The cross-reference listing can be suppressed by control card option.

Each new symbol encountered is entered in the SYMBOL column and its definition and/or cross-references are listed to the right through the TYPE, SEC, VALUE, DEFN, and REFERENCES columns. Each new symbol will have a SYMBOL and TYPE entry in the cross-reference listing. If the symbol is defined, the SEC, VALUE, and DEFN fields will be filled; if the symbol is not defined, these three fields will contain hyphens. A series of hyphens in any field indicates that, for that symbol, the field is not applicable, not available, or not known.

Symbol Column (SYMBOL): The SYMBOL column contains the symbol whose cross-references are listed in the succeeding columns and lines of the listing format. All entries prior to the next entry in the SYMBOL column refer to the given symbol.

```
                                        ALAMO CROSS-REFERENCE LISTING                    PAGE     2

 SYMBOL   TYPE SEC   VALUE    DEFN      REFERENCES                                        02/05/70

A         REL  01 00000008   684       676
ALPHA     REL  01 0000000E   690       678   679
A1        VAR  -- 00000011    29       671   672
B         REL  01 00000009   685       676
BETA      REL  01 0000000F   691       680   681
B1        VAR  -- 00000001    13       674
C         REL  01 0000000A   686       677
D         VAR  -- 00000000   678       682
E         REL  01 0000000B   687       677
F         REL  01 0000000C   688       682
G         VAR  -- 00000000   679       682
H         VAR  -- 00000000   680       683
I$N       ---  -- -------- UNDEF       273   276   320   337   354   371   412   415   505   508
J         VAR  -- 00000000   681       683
K         REL  01 0000000D   689       683
TST       ENT  01 00000000   675       674   692
X1        VAR  -- 00000021    45       677
X2        VAR  -- 00000022    46       683
 STEP-ASMC      DATE-02/05/70  EXECUTION TIME-00HRS 00MINS 40.08SECS
```

Figure 5-2. Cross-Reference Listing Example

Table 5-1. Assembler Generated Messages

FLAG	ERROR CONDITION
A	Addressability Error
B	Invalid Use of Base Intrinsic
C	Too Many Continuation Cards
D	Duplicate Label Assignment
E	General Syntax Error
F	Intrinsic Function Invalidly Used
G	Invalid Use of a List
H	Invalid Use of a Directive
I	Undefined Instruction
J	Invalid Use of Displacement Intrinsic
K	Invalid Use of Control Section Intrinsic
L	Error in the Label Field
M	Magnitude Error
N	No END Card on Deck
O	Too Many Operands on Statement
P	Parentheses Are Unbalanced
Q	Invalid Arithmetic Operation
R	Relocation Error
S	Truncation Has Occurred
T	Assembler Table Overflow
U	Undefined Symbol
V	Invalid Forward Reference
W	Warning, Possible Error
X	Reserved for Future Use
Y	Reserved for Future Use
Z	Disagreement in Location Counter between Pass One and Pass Two

Table 5-2. Procedure Processing Message Symbols

FLAG	ERROR CONDITION
1	Invalid use of a register, or insufficient number of parameters.
2	Questionable use of a register, or insufficient number of parameters.
3	Invalid use of a literal for the ASC.
4	Invalid use of an @ for the ASC.
5	Invalid use of a ¢ for the ASC.

Type Column (TYPE): The TYPE column contains an abbreviation specifying the type of the symbol being cross-referenced. The meanings of the abbreviations in the TYPE column are as follows:

ABBREVIATION	MEANING
REL	internally relocatable
ABS	absolute
EXT	external
ENT	entry point
VAR	variable, section and value may change; the first section and value is displayed

The VAR type symbol results from the definition of a SET or a DO directive outside of all procedures. Symbols defined by SET and DO directives inside a procedure have definition information only.

Section Column (SEC): The SEC column contains the number of the section to which the symbol belongs and in which it is defined.

Value Column (VALUE): The VALUE column contains the value assigned to or the evaluation of the symbol being cross-referenced. Depending upon the type of the symbol, its value may be an address constant, a value set by the programmer, or the evaluation of the symbol via operations. If the VALUE column contains eight asterisks, the value of the symbol cannot be represented in eight hexadecimal digits.

Definition Column (DEFN): The DEFN column contains the number of the statement in which the symbol is defined.

References Column (REFERENCES): The REFERENCES column contains a complete listing of the statement numbers of all the statements in which the symbol being cross-referenced appears. This will not include its definition statement number that appears in the DEFN column.

5-6. BINARY TEXT

The binary output (object module) from the Assembler is in card image form as input to the Link Editor. This output is the default option (see PROGRAMMER'S GUIDE TO THE JOB SPECIFICATION LANGUAGE).

Each control section is relocatable; instruction locations may be changed by the Link Editor.

SECTION VI
PROCEDURES

6-1. PROCEDURE BUILDING DIRECTIVES

All relocatable symbols defined within the procedure are defined as the same level and according to the same rules as symbols defined externally to the procedure. In general, then, the procedure definition should not contain symbols in the label fields (other than in PROC, NAME, DO, SNAME, and SET lines); otherwise, repeated invocations of the procedure would cause these symbols to have multiple definitions. However, labeling of lines of code within procedures is permitted by using intrinsic functions. Procedure definitions may be nested. Nested procedure definitions must be called by procedure reference lines, in order, from "outer nest" definition to "inner nest" definition. If a procedure reference line calls a procedure which is part of another procedure, and that other procedure has not been called, the inner procedure is undefined.

6-2. PROCEDURE DIRECTIVE (PROC)

The procedure is introduced by a PROC line and terminated by a PEND line.

The name of the procedure is stated in the label field of the PROC line. Values may be associated with a name by writing expressions in the operand field of the PROC line.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
symbol	ↀ	PROC	ↀ	[exp][, exp][, . . .]

Restrictions: The label must be present.

Code Field: The user has the ability to specify to the Assembler the number of words generated by a PROC directive in order to save assembly execution time. Consequently, columns 71 and 72 (see Topic 3-11) of the first card image in a

PROC directive are reserved as locations for a value that is the number of words generated, and if used for any other purpose, may lead to erroneous results. The value placed in columns 71 and 72 is right-justified in the field. The field should be used only when the number of words generated is fixed. The procedure, then, cannot optionally change the number of words generated (i.e., by calls to other procedures, or by containing a DO directive). The Assembler will give erroneous results if an erroneous value is given and will give an error flag (Z type error).

Example:

LABEL	COMMAND	OPERANDS
PROG1	PROC	A**B
	:	
	PEND	PROG1

A procedure named PROG1 is defined by a PROC directive. The value of the expression, A**B, is associated with the procedure named PROG1 and may be accessed by use of intrinsic functions.

6-3. PROCEDURE NAME DIRECTIVE (NAME)

Additional names for a procedure may be specified in the label fields of NAME directives.

Values may be associated with a name by writing expressions in the operand field. In the case of a multi-named procedure, values may be associated with each name.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
symbol	NAME	[exp][,exp][,...]

Restrictions: The label must be present.

Code Field: The user has the ability to specify to the Assembler the number of words generated by a NAME directive in order to save Assembly execution time. Consequently, columns 71 and 72 (see Topic 3-11) of the first card image in a

NAME directive are reserved as locations for a value that is the number of words generated, and if used for any other purpose, may lead to erroneous results. The value placed in columns 71 and 72 is right-justified in the field. The field should be used only when the number of words generated is fixed. The procedure, then, cannot optionally change the number of words generated (i.e., by calls to other procedures, or by containing a DO directive). The Assembler will give erroneous results if an erroneous value is given and will give an error flag (Z type error).

Example:

LABEL	COMMAND	OPERANDS
PROG2	PROC	A**B
BETA	NAME	A**C
GAMMA	NAME	A**D
	:	
	PEND	PROG2

A multi-named procedure is defined and may be referred to by the names PROG2, BETA, or GAMMA. The values of A**B, A**C, and A**D are associated with the names PROG2, BETA, and GAMMA, respectively.

6-4. PROCEDURE END DIRECTIVE (PEND)

The PEND directive terminates a procedure definition.

GENERAL FORM:

LABEL	COMMAND	OPERANDS
blank	PEND	symbol

Restrictions: For every PROC directive, there must be a PEND directive.

No label is allowed with the PEND directive.

The symbol in the operand must be the label of the PROC directive which initiated this procedure definition.

Examples:

LABEL	⌀	COMMAND	⌀	OPERANDS
INSTI		PROC		L, LI, M
		.		
		.		
		.		
		PEND		INSTI

The directive, PEND, terminates the definition of a procedure named INSTI.

6-5. STATEMENT NAME DIRECTIVE (SNAME)

The SNAME directive is used in the Assembler to assign an assembly control name to the next statement. It is used with the GOTO directive within procedure definitions to direct the Assembler to process procedure statements other than sequentially.

GENERAL FORM

LABEL		COMMAND		OPERANDS
symbol	⌀	SNAME	⌀	blank

Restrictions: The label of a SNAME statement is a local label and applies only to the procedure within which the SNAME statement appears; i.e., such a label could be defined differently by another SNAME statement in another procedure.

SNAME directives are illegal outside of procedures.

Examples:

LABEL	⌀	COMMAND	⌀	OPERANDS
TEST		PROC		
		.		
		.		
		.		
START		SNAME		
		B		INIT
		.		
		.		
		.		
		GOTO		START
		.		
		.		
		.		
		PEND		TEST

The label START of the SNAME statement is defined as the address of the next sequential statement. START is used as the operand of a GOTO statement within this same procedure and results in the assembly of the branch to INIT instruction immediately after the GOTO statement.

Restrictions: The label on the SNAME directives is not assigned a relocatable address by the assembler. It is only an assembly function and can be referenced only by a GOTO function.

6-6. GOTO DIRECTIVE (GOTO)

The GOTO directive is used within a procedure definition to transfer assembly control from the current statement.

Assembly control is transferred from the current statement to the statement indicated by the symbol in the operand of the GOTO statement.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	GOTO	↓	symbol

Restrictions: No label is allowed with the GOTO directive.

The symbol in the operand of the GOTO statement must be defined by a SNAME directive within the same procedure.

GOTO directives are illegal outside of procedures.

LABEL	↓	COMMAND	↓	OPERANDS
TEST		PROC		
		.		
		.		
		.		
START		SNAME		
		LI		A1, 10
		.		
		.		
		.		
		GOTO		START
		.		
		.		
		.		
		PEND		TEST

Control for this assembly is transferred to the address of the symbol START within the same procedure. START is defined by an SNAME directive within this procedure and results in the assembly of the load immediate instruction immediately after the GOTO statement.

6-7. PROCEDURE EXIT DIRECTIVE (PEX)

The PEX directive allows the user to exit immediately from the procedure in which it is used during expansion of the procedure, regardless of the location within the procedure definition of the PEX directive.

If expression is zero or blank, the exit will be out of all nested procedure references and the next statement will be processed from the main level assembly code. If expression is positive and absolute, the assembly will exit from that number of nested procedure references which is equal to the value of expression. If the value of expression is greater than the number of nested levels, the outermost level is selected.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	⊬	PEX	⊬	[exp]

Restrictions: No label is allowed with the PEX directive. PEX directives are illegal outside of procedures.

Limitations: The maximum value of expression is 32.

Examples:

LABEL	⊬	COMMAND	⊬	OPERANDS
		PEX		2

The directive will cause the assembly to exit from two levels of nested procedure calls.

6-8. FLAG DIRECTIVE (FLAG)

The FLAG directive causes a flag character to be printed on the left hand side of the listing if a specified condition is or is not met.

The character specified in the operand is printed to the left of the listing line if the value of expression is not zero.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	FLAG	↓	character,exp

Restrictions: Character must be a character string item of one character (enclosed in quotation marks).

Expression must be absolute.

No label is allowed with the FLAG directive. FLAG directives are not restricted to procedures.

Example:

LABEL	↓	COMMAND	↓	OPERANDS
		FLAG		"?", F ≥ 1000

The character, ?, will appear to the left of the listing each time the value of F is greater than or equal to 1000.

6-9. PROCEDURE EXAMPLES

The following examples of procedures are coded with the directives defined in Sections IV and VI and the machine instruction sets defined in Appendices G and H. Refer also to the programming manuals: PROGRAMMER'S GUIDE TO THE PERIPHERAL PROCESSOR and PROGRAMMER'S GUIDE TO THE CENTRAL PROCESSOR.

Example 1: The statements:

```
LABEL   COMMAND   OPERANDS
MOVE    PROC
        L         A1, VP(1)
        ST        A1, VP(2)
        PEND      MOVE
        :
        MOVE      AA, BB
        MOVE      CC, AA
```

produce code that moves the contents of AA to BB and the contents of CC to AA.

This procedure is in Central Processor code.

Example 2: Using the DO inside of a procedure is illustrated here:

```
LABEL   COMMAND   OPERANDS
SAVE    PROC
J       DO        I
        DATA      VP(1)
        PEND      SAVE
        :
I       SET       24
TAB     SAVE      0
I       SET       10
LISTQ   SAVE      I-3
```

to generate 24 words of zeros, followed by 10 words of the value 7. The labels are assigned to the first word of each block.

This procedure uses the Assembler facilities only and can be used in either Peripheral or Central Processor programs.

Example 3: The statements:

```
LABEL   COMMAND   OPERANDS
SETUP   PROC
        DO        FIT, 4, 3
        L         A1, VP(1)
        ST        A1, VP(2)
        L         A1, VP(1)+1
        ST        A1, VP(2)+1
        L         A1, = 0
        ST        A1, VP(2)
        ST        A1, VP(2)+1
        L         A1, = 0
        ST        A1, VP(2)+2
        PEND      SETUP
        .
        .
        .
FIT     SET       0
        SETUP     GROUPA, GROUPB
```

produce code that stores zeros in three words at GROUPB. If FIT were set to 1, the first two words of GROUPB would be filled with the two words from GROUPA. The third word of GROUPB would still be set to zero.

This procedure is in the Central Processor code.

Example 4: The statements:

LABEL	COMMAND	OPERANDS
MOVE	PROC	
J	DO	Z, 2
	L	A1, VP(1)-1+J
	ST	A1, VP(2)-1+J
	PEND	MOVE
BLOCK	PROC	
	DO	Z-5, 2
	L	A1, VP(1)
	ST	A1, VP(2)
K	DO	Z-2
	MOVE	VP(1)-4+4*K, VP(2)-4+4*K
	PEND	BLOCK
	:	
Z	SET	4
	BLOCK	ALPHA, OMEGA

Produce code that moves eight consecutive words from ALPHA to OMEGA.

This procedure is in Central Processor code.

Example 5: The statements:

```
LABEL   COMMAND   OPERANDS
MOVE    PROC      L, ST, A1
MOVEH   NAME      LH, STH, A2
I       DO        NP(0), 2
        RQ(1)     VQ(3), RP(I, 1)
        RQ (2)    VQ(3), RP(I, 2)
        PEND      MOVE
        .
        .
        .
        MOVE      (A, B), (C, (D, X2))
        MOVEH     (K, J), (L, M), (N, P)
```

gemerate the equivalent of the code:

```
LABEL   COMMAND   OPERANDS
        L         A1, A
        ST        A1, B
        L         A1, C
        ST        A1, D, X2
        LH        A2, K
        STH       A2, J
        LH        A2, L
        STH       A2, M
        LH        A2, N
        STH       A2, P
```

This procedure is in Central Processor code.

Example 6: The statements:

```
LABEL   COMMAND   OPERANDS
MOVE    PROC
        L         A1, RP(1)
        ST        A1, RP(2)
        PEND      MOVE
        .
        .
        .
        MOVE      A, B
        MOVE      (C, X1), E
        MOVE      ((D, G)), F
        MOVE      (@(H, J), X2), K
        .
        .
        .
```

generate the equivalent of the code:

```
LABEL   COMMAND   OPERANDS
        L         A1, A
        ST        A1, B
        L         A1, C, X1
        ST        A1, E
        L         A1, (D, G)
        ST        A1, F
        L         A1, @(H, J), X2
        ST        A1, K
```

This procedure is in Central Processor code.

Example 7: The statements:

```
LABEL   $  COMMAND   $  OPERANDS
HW$        FORM        16, 16
DATAHW     PROC
I          DO          NP(0)
           HW$         VP(I, 1), VP(I, 2)
           PEND        DATAHW
           .
           .
           .
           DATAHW      (1, 5), (2, 10), (3, -14)
           .
           .
           .
```

generate three words of halfword pairs and give a truncation indication on the sixth halfword.

This procedure uses only the Assembler facilities and can be used in either Peripheral or Central Processor programs.

Example 8: The statements:

```
LABEL   $  COMMAND   $  OPERANDS
MOVE       PROC        L, ST
TRANSFER   NAME        L, ST, ST, ST, ST, ST, ST, ST, ST
I          DO          NQ(0)
           CQ(I)       A1, VP(1)
           PEND        MOVE
           .
           .
           .
           MOVE        A, B
           TRANSFER    A, B, C, D, E, F, G, H, K
           .
           .
           .
```

generate the equivalent of the code:

LABEL	COMMAND	OPERANDS
	L	A1, A
	ST	A1, B
	L	A1, A
	ST	A1, B
	ST	A1, C
	ST	A1, D
	ST	A1, E
	ST	A1, F
	ST	A1, G
	ST	A1, H
	ST	A1, K

This procedure is in Central Processor code.

Example 9: To clear from A through B:

LABEL	COMMAND	OPERANDS
CLEAR	PROC	
I	SET	-1
	LI	A1, 0
AGAIN	SNAME	
I	SET	I+1
	ST	A1, VP(1)+I
	DO	(VP(1)+I < VP(2)
	GOTO	AGAIN
	PEND	CLEAR
	:	
	CLEAR	A, B
	:	

Note that A and B must be defined prior to use of this procedure. The procedure is in Central Processor code.

Example 10: Example 9 can also be coded:

```
LABEL  ¢  COMMAND  ¢  OPERANDS
CLEAR     PROC
          LI        A1, 0
I         DO        VP(2)-VP(1)+1
          ST        A1, VP(1)+I-1
          PEND      CLEAR
```

Note that A and B must be defined prior to use of this procedure.

Example 11: To clear B words beginning at location A:

```
LABEL  ¢  COMMAND  ¢  OPERANDS
CLEAR     PROC
          LI        A1, 0
I         DO        VP(2)
          ST        A1, VP(1)+I-1
          PEND      CLEAR
```

This procedure is in Central Processor code.

SECTION VII
MACHINE DEFINITION DIRECTIVES

7-1. GENERAL

The ASC Assembler will have all of its instructions defined by procedures. Examples in this volume will show how this can be done and how any instruction can be defined to the Assembler. The concept of procedure-oriented assemblers extends here to the realization that most assembly language code can be syntax-checked by the Assembler, if the right procedures are written, regardless of the object machine as long as the directive rules are followed. The procedures processed during Assembler initialization will contain not only instructions for the Central Processor and the Peripheral Processor, but vector instructions (aids to building vector parameters). The following directives are used to define machine characteristics which may be used to build instruction defining procedures. The machine characteristics are, in this Assembler, preprocessed to define the ASC language. However, the machine characteristics may be altered by using the extended directives, if their use precedes the use of any other directives, when the preprocessed set of procedures is not used.

7-2. UNIT DIRECTIVE (UNIT)

The UNIT directive is used to specify the number of bits in the object computer word. For word (byte) addressable memories, the UNIT directive specifies the word (byte) size. If the operand, expression, which must be absolute cannot be evaluated, or is evaluated as zero or negative, an error message is output on the listing. If no UNIT directive is furnished, the UNIT size will default to 32.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	⊬	UNIT	⊬	exp

Restrictions: No label is allowed with the UNIT directive.

Example:

LABEL		COMMAND		OPERANDS
	↓	UNIT	↓	24

7-3. SIZE DIRECTIVE (SIZE)

The SIZE directive is used to specify the number of bits required to represent the object computer address. References to relocatable quantities are permitted only within fields of length SIZE. An error is output on the assembly listing if the operand, expression, which must be absolute cannot be evaluated or is evaluated as zero or negative. If no SIZE is specified, it defaults to 24.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	SIZE	↓	exp

Restrictions: No label is allowed with the SIZE directive.

Example:

LABEL		COMMAND		OPERANDS
	↓	SIZE	↓	32

7-4. BASE DIRECTIVE (BASE)

The BASE directive is used to express the maximum number of bases allowed. Expression, which must be absolute, represents the maximum number of base registers. If expression is zero, then use of base and displacement addressing will not be allowed.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	BASE	↓	exp

Restrictions: No label is allowed with the BASE directive.

Example:

LABEL		COMMAND		OPERANDS
	↓	BASE	↓	31

7-5. DISP DIRECTIVE (DISP)

The DISP directive is used to specify the maximum size of a displacement field for base and displacement addressing. Expression must be positive and absolute. If no DISP directive is used, the displacement size defaults to 12.

GENERAL FORM:

LABEL		COMMAND		OPERANDS
blank	↓	DISP	↓	exp

Restrictions: No label is allowed with the DISP directive.

Example:

LABEL		COMMAND		OPERANDS
	↓	DISP	↓	12

7-6. EXAMPLES

An instruction set in directives for a hypothetical Central Processor could be defined using the following technique:

LABEL	↓	COMMAND	↓	OPERANDS
CPU		FORM		8, 4, 4, 4, 12
		⋮		
L		PROC		#11
LH		NAME		#12
LD		NAME		#13
		CPU		VQ(1), VP(1), IP(2)*8++VP(3), VP(2, 2)VP(2, 1)
		PEND		L

Using one of the above instructions, the code:

LABEL	∅	COMMAND	∅	OPERANDS
A10		EQU		10
		LH		A10,@(128,3),2

would create the following hexadecimal word:

1	2	A	A	3	0	8	0

APPENDIX A: EBCDIC CHARACTER SET

8-BIT CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
00000000	12,0,9,8,1		0	00
00000001	12,9,1		1	01
00000010	12,9,2		2	02
00000011	12,9,3		3	03
00000100	12,9,4		4	04
00000101	12,9,5		5	05
00000110	12,9,6		6	06
00000111	12,9,7		7	07
00001000	12,9,8		8	08
00001001	12,9,8,1		9	09
00001010	12,9,8,2		10	0A
00001011	12,9,8,3		11	0B
00001100	12,9,8,4		12	0C
00001101	12,9,8,5		13	0D
00001110	12,9,8,6		14	0E
00001111	12,9,8,7		15	0F
00010000	12,11,9,8,1		16	10
00010001	11,9,1		17	11
00010010	11,9,2		18	12
00010011	11,9,3		19	13
00010100	11,9,4		20	14
00010101	11,9,5		21	15
00010110	11,9,6		22	16
00010111	11,9,7		23	17
00011000	11,9,8		24	18
00011001	11,9,8,1		25	19
00011010	11,9,8,2		26	1A
00011011	11,9,8,3		27	1B
00011100	11,9,8,4		28	1C
00011101	11,9,8,5		29	1D
00011110	11,9,8,6		30	1E
00011111	11,9,8,7		31	1F
00100000	11,0,9,8,1		32	20
00100001	0,9,1		33	21
00100010	0,9,2		34	22
00100011	0,9,3		35	23
00100100	0,9,4		36	24
00100101	0,9,5		37	25
00100110	0,9,6		38	26
00100111	0,9,7		39	27
00101000	0,9,8		40	28

8-BIT CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
00101001	0,9,8,1		41	29
00101010	0,9,8,2		42	2A
00101011	0,9,8,3		43	2B
00101100	0,9,8,4		44	2C
00101101	0,9,8,5		45	2D
00101110	0,9,8,6		46	2E
00101111	0,9,8,7		47	2F
00110000	12,11,0,9,8,1		48	30
00110001	9,1		49	31
00110010	9,2		50	32
00110011	9,3		51	33
00110100	9,4		52	34
00110101	9,5		53	35
00110110	9,6		54	36
00110111	9,7		55	37
00111000	9,8		56	38
00111001	9,8,1		57	39
00111010	9,8,2		58	3A
00111011	9,8,3		59	3B
00111100	9,8,4		60	3C
00111101	9,8,5		61	3D
00111110	9,8,6		62	3E
00111111	9,8,7		63	3F
01000000		blank	64	40
01000001	12,0,9,1		65	41
01000010	12,0,9,2		66	42
01000011	12,0,9,3		67	43
01000100	12,0,9,4		68	44
01000101	12,0,9,5		69	45
01000110	12,0,9,6		70	46
01000111	12,0,9,7		71	47
01001000	12,0,9,8		72	48
01001001	12,8,1		73	49
01001010	12,8,2	¢	74	4A
01001011	12,8,3	. (period)	75	4B
01001100	12,8,4	<	76	4C
01001101	12,8,5	(	77	4D
01001110	12,8,6	+	78	4E
01001111	12,8,7	|	79	4F
01010000	12	&	80	50
01010001	12,11,9,1		81	51
01010010	12,11,9,2		82	52
01010011	12,11,9,3		83	53
01010100	12,11,9,4		84	54
01010101	12,11,9,5		85	55
01010110	12,11,9,6		86	56
01010111	12,11,9,7		87	57
01011000	12,11,9,8		88	58
01011001	11,8,1		89	59
01011010	11,8,2	!	90	5A
01011011	11,8,3	$	91	5B
01011100	11,8,4	*	92	5C
01011101	11,8,5	)	93	5D
01011110	11,8,6	;	94	5E
01011111	11,8,7	¬	95	5F
01100000	11	-	96	60
01100001	0,1	/	97	61
01100010	11,0,9,2		98	62
01100011	11,0,9,3		99	63
01100100	11,0,9,4		100	64

8-BIT CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
01100101	11,0,9,5		101	65
01100110	11,0,9,6		102	66
01100111	11,0,9.7		103	67
01101000	11,0,9,8		104	68
01101001	0,8,1		105	69
01101010	12,11		106	6A
01101011	0,8,3	,	107	6B
01101100	0,8,4	%	108	6C
01101101	0,8,5	_	109	6D
01101110	0,8,6	>	110	6E
01101111	0,8,7	?	111	6F
01110000	12,11,0		112	70
01110001	12,11,0,9,1		113	71
01110010	12,11,0,9,2		114	72
01110011	12,11,0,9,3		115	73
01110100	12,11,0,9,4		116	74
01110101	12,11,0,9,5		117	75
01110110	12,11,0,9,6		118	76
01110111	12,11,0,9,7		119	77
01111000	12,11,0,9,8		120	78
01111001	8,1	`	121	79
01111010	8,2	:	122	7A
01111011	8,3	#	123	7B
01111100	8.4	@	124	7C
01111101	8,5	' apostrophe	125	7D
01111110	8,6	=	126	7E
01111111	8,7	" (quotation marks)	127	7F
10000000	12,0,8,1		128	80
10000001	12,0,1		129	81
10000010	12,0,2		130	82
10000011	12,0,3		131	83
10000100	12,0,4		132	84
10000101	12,0,5		133	85
10000110	12,0,6		134	86
10000111	12,0,7		135	87
10001000	12,0,8		136	88
10001001	12,0,9		137	89
10001010	12,0,8,2		138	8A
10001011	12,0,8,3		139	8B
10001100	12,0,8,4		140	8C
10001101	12,0,8,5		141	8D
10001110	12,0,8,6		142	8E
10001111	12,0,8,7		143	8F
10010000	12,11,8,1		144	90
10010001	12,11,1		145	91
10010010	12,11,2		146	92
10010011	12,11,3		147	93
10010100	12,11,4		148	94
10010101	12,11,5		149	95
10010110	12,11,6		150	96
10010111	12,11,7		151	97
10011000	12,11,8		152	98
10011001	12,11,9		153	99
10011010	12,11,8,2		154	9A
10011011	12,11,8,3		155	9B
10011100	12,11,8,4		156	9C
10011101	12,11,8,5		157	9D
10011110	12,11,8,6		158	9E
10011111	12,11,8,7		159	9F
10100000	11,0,8,1		160	A0

8-BIT CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
10100001	11,0,1		161	A1
10100010	11,0,2		162	A2
10100011	11,0,3		163	A3
10100100	11,0,4		164	A4
10100101	11,0,5		16'	A5
10100110	11,0,6		166	A6
10100111	11,0,7		167	A7
10101000	11,0,8		168	A8
10101001	11,0,9		169	A9
10101010	11,0,8,2		170	AA
10101011	11,0,8,3		171	AB
10101100	11,0,8,4		172	AC
10101101	11,0,8,5		173	AD
10101110	11,0,8,6		174	AE
10101111	11,0,8,7		175	AF
10110000	12,11,0,8,1		176	B0
10110001	12,11,0,1		177	B1
10110010	12,11,0,2		178	B2
10110011	12,11,0,3		179	B3
10110100	12,11,0,4		180	B4
10110101	12,11,0,5		181	B5
10110110	12,11,0,6		182	B6
10110111	12,11,0,7		183	B7
10111000	12,11,0,8		184	B8
10111001	12,11,0,9		185	B9
10111010	12,11,0,8,2		186	BA
10111011	12,11,0,8,3		187	BB
10111100	12,11,0,8,4		188	BC
10111101	12,11,0,8,5		189	BD
10111110	12,11,0,8,6		190	BE
10111111	12,11,0,8,7		191	BF
11000000	12,0		192	C0
11000001	12,1	A	193	C1
11000010	12,2	B	194	C2
11000011	12,3	C	195	C3
11000100	12,4	D	196	C4
11000101	12,5	E	197	C5
11000110	12,6	F	198	C6
11000111	12,7	G	199	C7
11001000	12,8	H	200	C8
11001001	12,9	I	201	C9
11001010	12,0,9,8,2		202	CA
11001011	12,0,9,8,3		203	CB
11001100	12,0,9,8,4		204	CC
11001101	12,0,9,8,5		205	CD
11001110	12,0,9,8,6		206	CE
11001111	12,0,9,8,7		207	CF
11010000	11,0		208	D0
11010001	11,1	J	209	D1
11010010	11,2	K	210	D2
11010011	11,3	L	211	D3
11010100	11,4	M	212	D4
11010101	11,5	N	213	D5
11010110	11,6	O	214	D6
11010111	11,7	P	215	D7
11011000	11,8	Q	216	D8
11011001	11,9	R	217	D9
11011010	12,11,9,8,2		218	DA
11011011	12,11,9,8,3		219	DB
11011100	12,11,9,8,4		220	DC

8-BIT CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
11011101	12,11,9,8,5		221	DD
11011110	12,11,9,8,6		222	DE
11011111	12,11,9,8,7		223	DF
11100000	0,8,2		224	E0
11100001	11,0,9,1		225	E1
11100010	0,2	S	226	E2
11100011	0,3	T	227	E3
11100100	0,4	U	228	E4
11100101	0,5	V	229	E5
11100110	0,6	W	230	E6
11100111	0,7	X	231	E7
11101000	0,8	Y	232	E8
11101001	0,9	Z	233	E9
11101010	11,0,9,8,2		234	EA
11101011	11,0,9,8,3		235	EB
11101100	11,0,9,8,4		236	EC
11101101	11,0,9,8,5		237	ED
11101110	11,0,9,8,6		238	EE
11101111	11,0,9,8,7		239	EF
11110000	0	0	240	F0
11110001	1	1	241	F1
11110010	2	2	242	F2
11110011	3	3	243	F3
11110100	4	4	244	F4
11110101	5	5	245	F5
11110110	6	6	246	F6
11110111	7	7	247	F7
11111000	8	8	248	F8
11111001	9	9	249	F9
11111010	12,11,0,9,8,2		250	FA
11111011	12,11,0,9,8.3		251	FB
11111100	12,11,0,9,8,4		252	FC
11111101	12,11,0,9,8,5		253	FD
11111110	12,11,0,9,8,6		254	FE
11111111	12,11,0,9,8,7		255	FF

APPENDIX B: POWERS OF TWO TABLE

Table B-1. Powers of Two Table

2^{η}	η	$2^{-\eta}$
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
8 196	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 25
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	0.000 000 000 059 207 660 913 467 407 226 562 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	0.000 000 000 007 275 927 614 183 425 903 320 312 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

APPENDIX C: HEXADECIMAL ARITHMETIC TABLES

Table C-1. Hexadecimal Arithmetic Table

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Table C-2. Hexadecimal Multiplication Table

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

APPENDIX D: HEXADECIMAL TO DECIMAL CONVERSION TABLE

Table D-1. Hexadecimal to Decimal Conversion Table

The table in this appendix provides for direct conversion of decimal and hexadecimal numbers in these ranges:

Hexadecimal	Decimal
000 to FFF	0000 to 4095

For numbers outside the range of the table, add the following values to the table figures:

Hexadecimal	Decimal	Hexadecimal	Decimal
1000	4096	9000	36864
2000	8192	A000	40960
3000	12288	B000	45056
4000	16384	C000	49152
5000	20484	D000	53248
6000	24576	E000	57344
7000	28672	F000	61440
8000	32768		

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	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
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	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
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	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
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	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
[illegible]	[illegible]	3585	3586	3587	3583	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
[illegible]	[illegible]	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
[illegible]	[illegible]	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
[illegible]	[illegible]	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
[illegible]	[illegible]	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
[illegible]	[illegible]	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
[illegible]	[illegible]	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
[illegible]	[illegible]	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
[illegible]	[illegible]	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
[illegible]	[illegible]	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
[illegible]	[illegible]	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
[illegible]	[illegible]	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
[illegible]	[illegible]	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
[illegible]	[illegible]	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
[illegible]	[illegible]	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
[illegible]	[illegible]	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
[illegible]	[illegible]	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
[illegible]	[illegible]	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
[illegible]	[illegible]	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
[illegible]	[illegible]	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
[illegible]	[illegible]	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
[illegible]	[illegible]	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
[illegible]	[illegible]	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
[illegible]	[illegible]	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
[illegible]	[illegible]	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
[illegible]	[illegible]	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
[illegible]	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
[illegible]	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
[illegible]	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
[illegible]	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
[illegible]	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
[illegible]	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

APPENDIX E: OPERAND FORMATS

Table E-1: Operand Formats for the Central Processor

Type Lists	General Forms	Definitive Variations			
		Symbolic N		Explicit Base & Disp	
R, N, X Lists	r, [@] [=]n[, x]	r,@ =n, x r,@n, x r, =n, x r, n, x	r, @ =n r, @ n r, =n r, n	r,@ (d, b), x r, (d, b), x	r,@(d, b) r, (d, b)
	r, [@[=]]n[, x]	r,@ =n, x r,@n, x r, n, x	r,@=n r,@n r, n	r,@(d, b), x r, (d, b), x	r, @(d, b) r, (d, b)
	r, [@]n[, x]	r,@n, x r, n, x	r,@n r, n	r,@(d, b), x r, (d, b), x	r,@(d, b) r, (d, b)
	m, [@[=]n[, x]	m,@ =n, x m,@n, x m, n, x	m,@=n m, @n m, n	m, @(d, b), x m, (d, b), x	m, @(d, b) m, (d, b)
	m, [@]n[, x]	m,@n, x m, n, x	m,@n m, n	m, @(d, b), x m, (d, b), x	m, @(d, b) m, (d, b)
	[@[=]]n[, x]	@ =n, x @n, x n, x	@=n @n n	@(d, b), x (d, b), x	@(d, b) (d, b)
	[@]n[, x]	@n, x n, x	@n n	@(d, b), x (d, b), x	@(d, b) (d, b)
R, I, X Lists	r, i[, x] i[, x] i	r, i, x i, x	r, i i i	- - -	- - -
R, R, N Lists	r, r, n		r, r, n		r, r, (d, b)

Table E-2. Operand Formats for the Peripheral Processor

General Forms	Definitive Variations		General Forms	Definitive Variations	
	Indexed	Unindexed		Indexed	Unindexed
r,¢ [@][=]n[, x]	r,¢ @ =n, x r,¢ @ n, x r,¢ =n, x r,¢ n, x	r,¢ @ =n r,¢ @ n r,¢ =n r,¢ n	r, [@][=]n[, x]	r,@=n, x r,@ n, x r, =n, x r, n, x	r, @ =n r, @ n r, =n r, n
r,¢ [@]n[, x]	r,¢@ n, x r, ¢ n, x	r, ¢ @n r,¢ n	r, [@]n, x	r,@ n, x r, n, x	r, @n r, n
¢ [@[=]]n[, x]	¢@=n, x ¢@n, x ¢n, x	¢@ =n ¢@n ¢ n	[@[=]]n[, x]	@ =n, x @n, x n, x	@ =n @n n
¢ [@]n[, x]	¢@n, x ¢ n, x	¢ @n ¢n	[@]n[, x]	@n, x n, x	@n n
r, n[, x]	r, n, x	r, n	m, n[, x]	m, n, x	m, n
r, i[, x]	r, i, x	r, i	n[, x]	n, x	n

APPENDIX F: RESERVED REGISTER SYMBOLS

Table F-1. Reserved Peripheral Processor Register Symbols

Type	Number	Coding Symbol		
		Fullword	Halfword	Byte
VPR	4 (for each VP)	Rn	RnL, or RnR	RnBn
CR	64	Cm	CmL, or CmR	CmBn
Index	7 (the VPRs addressed by halfwords)	(none)	XnL, or XnR	(none)
where: $0 \leq n \leq 3$, and $0 \leq m \leq 63$				

Table F-2. Reserved Central Processor Register Symbols

Type	Number	Coding Symbol		
		Symbol	Decimal	Hexadecimal
BR	16	Bn	0 THRU 15	#0 THRU #F
AR	16	An	16 THRU 31	#10 THRU #1F
XR	8	Xm	32 THRU 39	#20 THRU #27
VR	8	Vm	40 THRU 47	#28 THRU #2F
where: $0 \leq n \leq 15$, and $0 \leq m \leq 7$				

APPENDIX G: PERIPHERAL PROCESSOR MACHINE INSTRUCTIONS

Table G-1. Peripheral Processor Instructions by Logical Grouping

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
ST	Store word from VPR to CM	14	r,[@]n[,x]	7-2
ST	Store word from VPR to CM augmented	1C	r,¢[@]n[,x]	7-2
ST	Store word from VPR to VPR	90	r,[@]n[,x]	7-2
ST	Store word from VPR to CR	98	r,[@]n[,x]	7-2
ST	Store word from CR to CM	10	r,[@]n[,x]	7-2
ST	Store word from CR to CM augmented	18	r,¢[@]n[,x]	7-2
ST	Store word from CR to VPR	94	r,[@]n[,x]	7-2
ST	Store word from CR to CR	9C	r,[@]n[,x]	7-2
STA	Store word from VPR to augmented CM absolute	1E	r,¢[@]n[,x]	7-3
STA	Store word from VPR to CM absolute	16	r,[@]n[,x]	7-3
STH	Store halfword from VPR to VPR	91	r,n[,x]	7-4
STH	Store halfword from VPR to CR	99	r,n[,x]	7-4
STH	Store halfword from CR to VPR	95	r,n[,x]	7-4
STH	Store halfword from CR to CR	9D	r,n[,x]	7-4
STB	Store byte from VPR to VPR	93	r,n[,x]	7-5
CTB	Store byte from VPR to CR	9B	r,n[,x]	7-5
STB	Store byte from CR to VPR	97	r,n[,x]	7-5
STB	Store byte from CR to CR	9F	r,n[,x]	7-5
STL	Store from VPR (left or right) to left half CM	15	r,[@]n[,x]	7-6
STL	Store from VPR to augmented left half of CM	1D	r,¢[@]n[,x]	7-6
STL	Store from CR to CM left half	11	r,[@]n[,x]	7-6
STL	Store from CR to augmented CM left half	19	r,¢[@]n[,x]	7-6
STR	Store from VPR to CM right half	17	r,[@]n[,x]	7-7
STR	Store from VPR to augmented CM right half	1F	r,¢[@]n[,x]	7-7
STR	Store from CR to CM right half	13	r,[@]n[,x]	7-7
STR	Store from CR to augmented CM right half	1B	r,¢[@]n[,x]	7-7
STF	Store file from VPR into augmented CM	1A	¢[@]n[,x]	7-8
STF	Store file from VPR into CM	2A	[@]n[,x]	7-8
LD	Load word to VPR from CM	04	r,[@][=]n[,x]	7-10
LD	Load word to VPR from augmented CM	0C	r,¢[@][=]n[,x]	7-10
LD	Load word to VPR from VPR	80	r,[@]n[,x]	7-10
LD	Load word to VPR from CR	88	r,[@]n[,x]	7-10
LD	Load word to CR from CM	38	r,[@][=]n[,x]	7-10
LD	Load word to CR from augmented CM	08	r,¢[@][=]n[,x]	7-10
LD	Load word to CR from VPR	84	r,[@]n[,x]	7-10
LD	Load word to CR from CR	83	r,[@]n[,x]	7-10

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
LDA	Load word to VPR from CM absolute	06	r,[@][=]n[,x]	7-11
LDA	Load word to VPR from augmented CM absolute	0E	r,¢[@][=]n[,x]	7-11
LDI	Load immediate word into VPR	72	r,i[,x]	7-12
LDI	Load immediate word into CR	62	r,i[,x]	7-12
LDH	Load halfword to VPR from VPR	81	r,n[,x]	7-13
LDH	Load halfword to VPR from CR	89	r,n[,x]	7-13
LDH	Load halfword to CR from VPR	85	r,n[,x]	7-13
LDH	Load halfword to CR from CR	8D	r,n[,x]	7-13
LDHI	Load immediate halfword into VPR	76	r,i[,x]	7-14
LDHI	Load immediate halfword into CR	66	r,i[,x]	7-14
LDB	Load byte to VPR from VPR	83	r,n[,x]	7-15
LDB	Load byte to VPR from CR	8B	r,n[,x]	7-15
LDB	Load byte to CR from VPR	87	r,n[,x]	7-15
LDB	Load byte to CR from CR	8F	r,n[,x]	7-15
LDBI	Load immediate byte into VPR	7E	r,i[,x]	7-16
LDBI	Load immediate byte into CR	6E	r,i[,x]	7-16
LDL	Load to VPR from CM left half	05	r,[@][=]n[,x]	7-17
LDL	Load to VPR from CM augmented left half	0D	r,¢[@][=]n[,x]	7-17
LDL	Load to CR from CM left half	39	r,[@][=]n[,x]	7-17
LDL	Load to CR from CM augmented left half	09	r,¢[@][=]n[,x]	7-17
LDR	Load to VPR from CM right half	07	r,[@][=]n[,x]	7-18
LDR	Load to VPR from augmented CM right half	0F	r,¢[@][=]n[,x]	7-18
LDR	Load to CR from CM right half	3B	r,[@][=]n[,x]	7-18
LDR	Load to CR from CM augmented right half	0B	r,¢[@][=]n[,x]	7-18
LDF	Load file from CM augmented into VPR	0A	¢[@]n[,x]	7-19
LDF	Load file from CM into VPR	3A	[@]n[,x]	7-19
AD	Add word in CM to VPR	50	r,[@][=]n[,x]	7-21
AD	Add word in VPR to VPR	D0	r,[@]n[,x]	7-21
ADI	Add immediate word to VPR	70	r,i[,x]	7-22
ADH	Add halfword in VPR to VPR	D1	r,n[,x]	7-23
ADHI	Add immediate halfword to VPR	71	r,i[,x]	7-24
ADB	Add byte in VPR to VPR	D3	r,n[,x]	7-25
ADBI	Add immediate byte to VPR	73	r,i[,x]	7-26

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
ADL	Add left half in CM to VPR	51	r,[@][=]n[,x]	7-27
ADR	Add right half in CM to VPR	53	r,[@][=]n[,x]	7-28
SU	Subtract word in CM from VPR	54	r,[@][=]n[,x]	7-29
SU	Subtract word in VPR from VPR	D4	r,[@]n[,x]	7-29
SUI	Subtract immediate word from VPR	74	r,i[,x]	7-30
SUH	Subtract halfword in VPR from VPR	D5	r,n[,x]	7-31
SUHI	Subtract immediate halfword from VPR	75	r,i[,x]	7-32
SUB	Subtract byte in VPR from VPR	D7	r,n[,x]	7-33
SUBI	Subtract immediate byte from VPR	77	r,i[,x]	7-34
SUL	Subtract left half in CM from VPR	55	r,[@][=]n[,x]	7-35
SUR	Subtract right half in CM from VPR	57	r,[@][=]n[,x]	7-36
OR	Logical OR word in CM to VPR	44	r,[@][=]n[,x]	7-39
OR	Logical OR word in VPR to VPR	C4	r,[@]n[,x]	7-39
OR	Logical OR word in CR to VPR	E4	r,[@]n[,x]	7-39
ORH	Logical OR halfword in VPR to VPR	C5	r,n[,x]	7-39
ORH	Logical OR halfword in CR to VPR	E5	r,n[,x]	7-40
ORHI	Logical OR immediate halfword to VPR	65	r,i[,x]	7-41
ORB	Logical OR byte in VPR to VPR	C7	r,n[,x]	7-42
ORB	Logical OR byte in CR to VPR	E7	r,n[,x]	7-42
ORBI	Logical OR immediate byte to VPR	67	r,i[,x]	7-43
ORL	Logical OR left half in CM to VPR	45	r,[@][=]n[,x]	7-44
ORR	Logical OR right half in CM to VPR	47	r,[@][=]n[,x]	7-45
EX	Logical exclusive OR word in CM to VPR	4C	r,[@][=]n[,x]	7-46
EX	Logical exclusive OR word in VPR to VPR	CC	r,[@]n[,x]	7-46
EX	Logical exclusive OR word in CR to VPR	EC	r,[@]n[,x]	7-46
EXH	Logical exclusive OR halfword in VPR to VPR	CD	r,n[,x]	7-47
EXH	Logical exclusive OR halfword in CR to VPR	ED	r,n[,x]	7-47
EXHI	Logical exclusive OR immediate halfword to VPR	6D	r,i[,x]	7-48
EXB	Logical exclusive OR byte in VPR to VPR	CF	r,n[,x]	7-49
EXB	Logical exclusive OR byte in CR to VPR	EF	r,n[,x]	7-49
EXBI	Logical exclusive OR immediate byte to VPR	6F	r,i[,x]	7-50
EXL	Logical exclusive OR left half CM to VPR	4D	r,[@][=]n[,x]	7-51

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
EXR	Logical exclusive OR right half CM to VPR	4F	r,[@][=]n[,x]	7-52
AN	Logical AND word in CM to VPR	40	r,[@][=]n[,x]	7-54
AN	Logical AND word in VPR to VPR	C0	r,[@]n[,x]	7-54
AN	Logical AND word in CR to VPR	E0	r,[@]n[,x]	7-54
ANH	Logical AND halfword in VPR to VPR	C1	r,n[,x]	7-55
ANH	Logical AND halfword in CR to VPR	E1	r,n[,x]	7-55
ANHI	Logical AND immediate halfword to VPR	61	r,i[,x]	7-56
ANB	Logical AND byte in VPR to VPR	C3	r,n[,x]	7-57
ANB	Logical AND byte in CR to VPR	E3	r,n[,x]	7-57
ANBI	Logical AND immediate byte to VPR	63	r,i[,x]	7-58
ANL	Logical AND left half in CM to VPR	41	r,[@][=]n[,x]	7-59
ANR	Logical AND right half in CM to VPR	43	r,[@][=]n[,x]	7-60
EQ	Logical EQUIVALENCE word CM to VPR	48	r,[@][=]n[,x]	7-62
EQ	Logical EQUIVALENCE word VPR to VPR	C8	r,[@]n[,x]	7-62
EQ	Logical EQUIVALENCE word CR to VPR	E8	r,[@]n[,x]	7-62
EQH	Logical EQUIVALENCE halfword VPR to VPR	C9	r,n[,x]	7-63
EQH	Logical EQUIVALENCE halfword CR to VPR	E9	r,n[,x]	7-63
EQHI	Logical EQUIVALENCE immediate halfword to VPR	69	r,i[,x]	7-64
EQB	Logical EQUIVALENCE byte VPR to VPR	CB	r,n[,x]	7-65
EQB	Logical EQUIVALENCE byte CR to VPR	EB	r,n[,x]	7-65
EQBI	Logical EQUIVALENCE immediate byte to VPR	6B	r,i[,x]	7-66
EQL	Logical EQUIVALENCE left half CM to VPR	49	r,[@][=]n[,x]	7-67
EQR	Logical EQUIVALENCE right half CM to VPR	4B	r,[@][=]n[,x]	7-68
CE	Compare word CM to VPR, skip if equal	30	r,[@][=]n[,x]	7-70
CE	Compare word VPR to VPR, skip if equal	D8	r,[@]n[,x]	7-70
CE	Compare word CR to VPR, skip if equal	F8	r,[@]n[,x]	7-70
CEI	Compare immediate word with VPR, skip if equal	78	r,i[,x]	7-71
CEH	Compare halfword VPR to VPR, skip if equal	D9	r,n[,x]	7-72
CEH	Compare halfword CR to VPR, skip if equal	F9	r,n[,x]	7-72
CEHI	Compare immediate halfword with VPR, skip if equal	79	r,i[,x]	7-73
CEB	Compare byte VPR to VPR, skip if equal	DB	r,n[,x]	7-73

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
CEB	Compare byte CR to VPR, skip if equal	FB	r, n[, x]	7-74
CEBI	Compare immediate byte with VPR, skip if equal	7B	r, i[, x]	7-75
CEL	Compare left half of CM to VPR, skip if equal	31	r, [@][=]n[, x]	7-76
CER	Compare right half CM to VPR, skip if equal	33	r, [@][=]n[, x]	7-77
CN	Compare word CM to VPR, skip if not equal	34	r, [@][=]n[, x]	7-78
CN	Compare word VPR to VPR, skip if not equal	DC	r, [@]n[, x]	7-78
CN	Compare word CR to VPR, skip if not equal	FC	r, [@]n[, x]	7-78
CNI	Compare immediate word with VPR, skip if not equal	7C	r, i[, x]	7-79
CNH	Compare halfword VPR to VPR, skip if not equal	DD	r, n[, x]	7-80
CNH	Compare halfword CR to VPR, skip if not equal	FD	r, n[, x]	7-80
CNHI	Compare immediate halfword with VPR, skip if not equal	7D	r, i[, x]	7-81
CNB	Compare byte VPR to VPR, skip if not equal	DF	r, n[, x]	7-82
CNB	Compare byte CR to VPR, skip if not equal	FF	r, n, [, x]	7-82
CNBI	Compare immediate byte with VPR, skip if not equal	7F	r, i[, x]	7-83
CNL	Compare left half CM to VPR, skip if not equal	35	r, [@][=]n[, x]	7-84
CNR	Compare right half CM to VPR, skip if not equal	37	r, [@][=]n[, x]	7-85
BC	Branch unconditionally to CM, base relative	02	[@]n[, x]	7-91
BC	Branch unconditionally to augmented CM, base relative	32	¢[@]n[, x]	7-91
BCS	Branch unconditionally to CM relative base, save PC in VPR	12	r, [@][=]n[, x]	7-87
BCA	Branch unconditionally to absolute CM	4A	[@[=]]n[, x]	7-88
BCA	Branch unconditionally to augmented absolute CM	4E	¢[@]n[, x]	7-88
BCAS	Branch unconditionally to CM absolute, save PC in VPR	52	r, [@][=]n[, x]	7-89
BPC	Branch unconditionally to CM, relative PC	5A	[@]n[, x]	7-90
BPC	Branch unconditionally to augmented CM, relative PC	5E	¢[@]n[, x]	7-90
BPCS	Branch unconditionally to CM, relative PC, save PC in VPR	AE	r, [@][=]n[, x]	7-92
BR	Branch unconditionally to ROM	42	[@]n[, x]	7-93

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
BRS	Branch unconditionally to ROM, save PC in VPR	46	r,[@][=]n[,x]	7-94
BRSM	Branch unconditionally to absolute ROM, save PC in CM location 20_{16}, augmented	56	n[,x]	7-95
PUSH	Push word from VPR into stack	58	r,[@]n[,x]	7-97
PULL	Pull word from stack into VPR	59	r,[@]n[,x]	7-98
MOD	Modify stack by contents of VPR halfword	5B	r,[@]n[,x]	7-99
VPS	Set VP flag in CR	86	n[,x]	7-101
VPR	Reset VP flag in CR	82	r[,x]	7-102
VPTO	Test VP flag in CR, skip if equal to one	8E	n[,x]	7-103
VPTZ	Test VP flag in CR, skip if equal to zero	8A	n[,x]	7-104
SL	Set mask bits in left half of CR byte	FA	m,n[,x]	7-106
SR	Set mask bits in right half of CR byte	FE	m,n[,x]	7-106
RL	Reset mask bits in left half of CR byte	F2	m,n[,x]	7-107
RR	Reset mask bits in right half of CR byte	F6	m,n[,x]	7-107
TOL	Test under mask for any ones in left half of CR byte and skip if true	CA	m,n[,x]	7-109
TOR	Test under mask for any ones in right half of CR byte and skip if true	CE	m,n[,x]	7-109
TZL	Test under mask for any zeros in left half of CR byte and skip if true	CZ	m,n[,x]	7-110
TZR	Test under mask for any zeros in right half of CR byte and skip if true	C6	m,n[,x]	7-110
TAOL	Test under mask for all ones in left half of CR byte and skip if true	EA	m,n[,x]	7-111
TAOR	Test under mask for all ones in right half of CR byte and skip if true	EE	m,n[,x]	7-111
TAZL	Test under mask for all zeros in left half of CR byte and skip if true	E2	m,n[,x]	7-112
TAZR	Test under mask for all zeros in right half of CR byte and skip if true	E6	m,n[,x]	7-112
SHL	Shift logical word in VPR	64	r,i[,x]	7-114
SHA	Shift arithmetic word in VPR	60	r,i[,x]	7-115
SHC	Shift cyclic word in VPR	6C	r,i[,x]	7-116
TSZL	Test under mask for any zeros in left half of CR byte and set; then skip if true	D2	m,n[,x]	7-118

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
TSZR	Test under mask for any zeros in right half of CR byte and set; then skip if true	D6	m, n[, x]	7-118
TSOL	Test under mask for any ones in left half of CR byte and set; then skip if true	DA	m, n[, x]	7-119
TSOR	Test under mask for any ones in right half of CR byte and set; then skip if true	DE	m, n[, x]	7-119
TRZL	Test under mask for any zeros in left half of CR byte and reset; then skip if true	92	m, n[, x]	7-120
TRZR	Test under mask for any zeros in right half of CR byte and reset; then skip if true	96	m, n[, x]	7-120
TROL	Test under mask for any ones in left half of CR byte and reset; then skip if true	9A	m, n[, x]	7-121
TROR	Test under mask for any ones in right half of CR byte and reset; then skip if true	9E	m, n[, x]	7-121
IBZ	Increment VPR by one; branch if result equals zero	B2	r, [@][=]n[, x]	7-123
IBN	Increment VPR by one; branch if result is not equal to zero	B6	r, [@][=]n[, x]	7-123
DBZ	Decrement VPR by one; branch if result equals zero	BA	r, [@][=]n[, x]	7-124
DBN	Decrement VPR by one; branch if result is not equal to zero	BF	r, [@][=]n[, x]	7-124
TZ	Test VPR word arithmetically; branch if equal to zero	B0	r, [@][=]n[, x]	7-126
TZ	Test CR word arithmetically; branch if equal to zero	A0	r, [@][=]n[, x]	7-126
TZH	Test VPR halfword arithmetically; branch if equal to zero	B1	r, [@][=]n[, x]	7-126
TZH	Test CR halfword arithmetically; branch if equal to zero	A1	r, [@][=]n[, x]	7-126
TZB	Test VPR byte arithmetically; branch if equal to zero	B3	r, [@][=]n[, x]	7-126
TZB	Test CR byte arithmetically; branch if equal to zero	A3	r, [@][=]n[, x]	7-126
TN	Test VPR word arithmetically; branch if not equal to zero	B4	r, [@][=]n[, x]	7-127
TN	Test CR word arithmetically; branch if not equal to zero	A4	r, [@][=]n[, x]	7-127
TNH	Test VPR halfword arithmetically; branch if not equal to zero	B5	r, [@][=]n[, x]	7-127

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
TNH	Test CR halfword arithmetically; branch if not equal to zero	A5	r,[@][=]n[,x]	7-127
TNB	Test VPR byte arithmetically; branch if not equal to zero	B7	r,[@][=]n[,x]	7-127
TNB	Test CR byte arithmetically; branch if not equal to zero	A7	r,[@][=]n[,x]	7-127
TP	Test VPR word arithmetically; branch if greater than or equal to zero	B8	r,[@][=]n[,x]	7-128
TP	Test CR word arithmetically; branch if greater than or equal to zero	A8	r,[@][=]n[,x]	7-128
TPH	Test VPR halfword arithmetically; branch if greater than or equal to zero	B9	r,[@][=]n[,x]	7-128
TPH	Test CR halfword arithmetically; branch if greater than or equal to zero	A9	r,[@][=]n[,x]	7-128
TPB	Test VPR byte arithmetically; branch if greater than or equal to zero	BB	r,[@][=]n[,x]	7-128
TPB	Test CR byte arithmetically; branch if greater than or equal to zero	AB	r,[@][=]n[,x]	7-128
TM	Test VPR word arithmetically; branch if less than zero	BC	r,[@][=]n[,x]	7-129
TM	Test CR word arithmetically; branch if less than zero	AC	r,[@][=]n[,x]	7-129
TMH	Test VPR halfword arithmetically; branch if less than zero	BD	r,[@][=]n[,x]	7-129
TMH	Test CR halfword arithmetically; branch if less than zero	AD	r,[@][=]n[,x]	7-129
TMB	Test VPR byte arithmetically; branch if less than zero	BF	r,[@][=]n[,x]	7-129
TMB	Test CR byte arithmetically; branch if less than zero	AF	r,[@][=]n[,x]	7-129
LDEA	Load effective address into VPR	5D	r,[@][=]n[,x]	7-131
ANAZ	Analyze CM	5F	r,[@][=]n[,x]	7-132
POLL	Poll CR and set VPR	F5	r,n[,x]	7-133
EXEC	Execute CM	5C	[@[=]]n[,x]	7-134
LDMB	Load VP base from VPR to CR	F4	[@]n[,x]	7-135
NOP	No operation	00	r,[@]n[,x]	7-136

Table G-2. Peripheral Processor Instructions In Alphabetical Order by Assembler Code

ASSMB CODE	MCHN CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
AD	50	Add word in CM to VPR	r,[@][=]n[,x]	7-21
AD	D0	Add word in VPR to VPR	r,[@]n[,x]	7-21
ADB	D3	Add byte in VPR to VPR	r,n[,x]	7-25
ADBI	73	Add immediate byte to VPR	r,i[,x]	7-26
ADH	D1	Add halfword in VPR to VPR	r,n[,x]	7-23
ADHI	71	Add immediate halfword to VPR	r,i[,x]	7-24
ADI	70	Add immediate word to VPR	r,i[,x]	7-22
ADL	51	Add left half in CM to VPR	r,[@][=]n[,x]	7-27
ADR	53	Add right half in CM to VPR	r,[@][=]n[,x]	7-28
AN	40	Logical AND word in CM to VPR	r,[@][=]n[,x]	7-54
AN	C0	Logical AND word in VPR to VPR	r,[@]n[,x]	7-54
AN	E0	Logical AND word in CR to VPR	r,[@]n[,x]	7-54
ANAZ	5F	Analyze CM	r,[@][=]n[,x]	7-132
ANB	C3	Logical AND byte in VPR to VPR	r,n[,x]	7-57
ANB	E3	Logical AND byte in CR to VPR	r,n[,x]	7-57
ANBI	63	Logical AND immediate byte to VPR	r,i[,x]	7-58
ANH	C1	Logical AND halfword in VPR to VPR	r,n[,x]	7-55
ANH	E1	Logical AND halfword in CR to VPR	r,n[,x]	7-55
ANHI	61	Logical AND immediate halfword to VPR	r,i[,x]	7-56
ANL	41	Logical AND left half in CM to VPR	r,[@][=]n[,x]	7-59
ANR	43	Logical AND right half in CM to VPR	5,[@][=]n[,x]	7-60
BC	02	Branch unconditionally to CM, base relative	[@]n[,x]	7-91
BC	32	Branch unconditionally to augmented CM, base relative	¢[@]n[,x]	7-91
BCA	4A	Branch unconditionally to absolute CM	[@[=]]n[,x]	7-88
BCA	4E	Branch unconditionally to augmented absolute CM	¢[@[=]]n[,x]	7-88
BCAS	52	Branch unconditionally to CM absolute, save PC in VPR	r,[@][=]n[,x]	7-89
BCS	12	Branch unconditionally to CM relative base, save PC in VPR	r,[@][=]n[,x]	7-87
BPC	5A	Branch unconditionally to CM, relative PC	[@]n[,x]	7-90
BPC	5E	Branch unconditionally to augmented CM, relative PC	¢[@]n[,x]	7-90
BPCS	AE	Branch unconditionally to CM, relative PC, save PC in VPR	r,[@][=]n[,x]	7-92

ASSMB CODE	MCHN CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
BR	42	Branch unconditionally to ROM	[@]n[,x]	7-93
BRS	46	Branch unconditionally to ROM, save PC in VPR	r,[@][=]n[,x]	7-94
BRSM	56	Branch unconditionally to absolute ROM. Save PC in CM location 20_{16}, augmented	n[,x]	7-95
CE	30	Compare word CM to VPR, skip if equal	r,[@][=]n[,x]	7-70
CE	D8	Compare word VPR to VPR, skip if equal	r,[@]n[,x]	7-70
CE	F8	Compare word CR to VPR, skip if equal	r,[@]n[,x]	7-70
CEB	DB	Compare byte VPR to VPR, skip if equal	r,n[,x]	7-73
CEB	FB	Compare byte CR to VPR, skip if equal	r,n[,x]	7-73
CEBI	7B	Compare immediate byte with VPR, skip if equal	r,i[,x]	7-75
CEH	D9	Compare halfword VPR to VPR, skip if equal	r,n[,x]	7-72
CEH	F9	Compare halfword CR to VPR, skip if equal	r,n[,x]	7-72
CEHI	79	Compare immediate halfword with VPR, skip if equal	r,i[,x]	7-73
CEI	78	Compare immediate word with VPR, skip if equal	r,i[,x]	7-71
CEL	31	Compare left half CM to VPR, skip if equal	r,[@][=]n[,x]	7-76
CER	33	Compare right half CM to VPR, skip if equal	r,[@][=]n[,x]	7-77
CN	34	Compare word CM to VPR, skip if not equal	r,[@][=]n[,x]	7-78
CN	DC	Compare word VPR to VPR, skip if not equal	r,[@]n[,x]	7-78
CN	FC	Compare word CR to VPR, skip if not equal	r,[@]n[,x]	7-78
CNB	DF	Compare byte VPR to VPR, skip if not equal	r,n[,x]	7-82
CNB	FF	Compare byte CR to VPR, skip if not equal	r,n[,x]	7-82
CNBI	7F	Compare immediate byte with VPR, skip if not equal	r,i[,x]	7-83
CNH	DD	Compare halfword VPR to VPR, skip if not equal	r,n[,x]	7-80
CNH	FD	Compare halfword CR to VPR, skip if not equal	r,n[,x]	7-80
CNHI	7D	Compare immediate halfword with VPR, skip if not equal	r,i[,x]	7-81
CNI	7C	Compare immediate word with VPR, skip if not equal	r,i[,x]	7-79
CNL	35	Compare left half CM to VPR, skip if not equal	r,[@][=]n[,x]	7-84
CNR	37	Compare right half CM to VPR, skip if not equal	r,[@][=]n[,x]	7-85
DBN	BE	Decrement VPR by one; branch if result is not equal to zero	r,[@][=]n[,x]	7-124

ASSMB CODE	MCHN CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
DBZ	BA	Decrement VPR by one; branch if result equals zero	r,[@][=]n[,x]	7-124
EQ	48	Logical EQUIVALENCE word CM to VPR	r,[@][=]n[,x]	7-62
EQ	C8	Logical EQUIVALENCE word VPR to VPR	r,[@]n[,x]	7-62
EQ	E8	Logical EQUIVALENCE word CR to VPR	r,[@]n[,x]	7-62
EQB	CB	Logical EQUIVALENCE byte VPR to VPR	r,n[,x]	7-65
EQB	EB	Logical EQUIVALENCE byte CR to VPR	r,n[,x]	7-65
EQBI	6B	Logical EQUIVALENCE immediate byte to VPR	r,i[,x]	7-66
EQH	C9	Logical EQUIVALENCE halfword VPR to VPR	r,n[,x]	7-63
EQH	E9	Logical EQUIVALENCE halfword CR to VPR	r,n[,x]	7-63
EQHI	69	Logical EQUIVALENCE immediate halfword to VPR	r,i[,x]	7-64
EQL	49	Logical EQUIVALENCE left half CM to VPR	r,[@][=]n[,x]	7-67
EQR	4B	Logical EQUIVALENCE right half CM to VPR	r,[@][=]n[,x]	7-68
EX	4C	Logical exclusive OR word in CM to VPR	r,[@][=]n[,x]	7-46
EX	CC	Logical exclusive OR word in VPR to VPR	r,[@]n[,x]	7-46
EX	EC	Logical exclusive OR word in CR to VPR	r,[@]n[,x]	7-46
EXB	CF	Logical exclusive OR byte in VPR to VPR	r,n[,x]	7-49
EXB	EF	Logical exclusive OR byte in CR to VPR	r,n[,x]	7-49
EXBI	6F	Logical exclusive OR immediate byte to VPR	r,i[,x]	7-50
EXEC	5C	Execute CM	[@[=]]n[,x]	7-134
EXH	CD	Logical exclusive OR halfword in VPR to VPR	r,n[,x]	
EXH	ED	Logical exclusive OR halfword in CR to VPR	r,n[,x]	7-47
EXHI	6D	Logical exclusive OR immediate halfword to VPR	r,i[,x]	7-48
EXL	4D	Logical exclusive OR left half CM to VPR	r,[@][=]n[,x]	7-51
EXR	4F	Logical exclusive OR right half CM to VPR	r,[@][=]n[,x]	7-52
IBN	B6	Increment VPR by one; branch if result is not equal to zero	r,[@][=]n[,x]	7-123
IBZ	B2	Increment VPR by one; branch if result equals zero	r,[@][=]n[,x]	7-123
LD	04	Load word to VPR from CM	r,[@][=]n[,x]	7-10
LD	0C	Load word to VPR from augmented CM	r,¢[@][=]n[,x]	7-10
LD	80	Load word to VPR from VPR	r,[@]n[,x]	7-10
LD	88	Load word to VPR from CR	r,[@]n[,x]	7-10

ASSMB CODE	MCHN CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
LD	38	Load word to CR from CM	r,[@][=]n[,x]	7-10
LD	08	Load word to CR from augmented CM	r,¢[@][=]n[,x]	7-10
LD	84	Load word to CR from VPR	r,[@]n[,x]	7-10
LD	8C	Load word to CR from CR	r,[@]n[,x]	7-10
LDA	06	Load word to VPR from CM absolute	r,[@][=]n[,x]	7-11
LDA	0E	Load word to VPR from augmented CM	r,¢[@][=]n[,x]	7-11
LDB	83	Load byte to VPR from VPR	r,n[,x]	7-15
LDB	8B	Load byte to VPR from CR	r,n[,x]	7-15
LDB	87	Load byte to CR from VPR	r,n[,x]	7-15
LDB	8F	Load byte to CR from CR	r,n[,x]	7-15
LDBI	7E	Load immediate byte into VPR	r,i[,x]	7-16
LDBI	6E	Load immediate byte into CR	r,i[,x]	7-16
LDEA	5D	Load effective address into VPR	r,[@][=]n[,x]	7-131
LDF	0A	Load file from CM augmented into VPR	¢[@]n[,x]	7-19
LDF	3A	Load file from CM into VPR	[@]n[,x]	7-19
LDH	81	Load halfword to VPR from VPR	r,n[,x]	7-13
LDH	89	Load halfword to VPR from CR	r,n[,x]	7-13
LDH	85	Load halfword to CR from VPR	r,n[,x]	7-13
LDH	8D	Load halfword to CR from CR	r,n[,x]	7-13
LDHI	76	Load immediate halfword into VPR	r,i[,x]	7-14
LDHI	66	Load immediate halfword into CR	r,i[,x]	7-14
LDI	72	Load immediate word into VPR	r,i[,x]	7-12
LDI	62	Load immediate word into CR	r,i[,x]	7-12
LDL	05	Load to VPR from CM left half	r,[@][=]n[,x]	7-17
LDL	0D	Load to VPR from CM augmented left half	r,¢[@][=]n[,x]	7-17
LDL	39	Load to CR from CM left half	r,[@][=]n[,x]	7-17
LDL	09	Load to CR from CM augmented left half	r,¢[@][=]n[,x]	7-17
LDMB	F4	Load VP base from VPR to CR	[@]n[,x]	7-135
LDR	07	Load to VPR from CM right half	r,[@][=]n[,x]	7-18
LDR	0F	Load to VPR from augmented CM right half	r,¢[@][=]n[,x]	7-18
LDR	3B	Load to CR from CM right half	r,[@][=]n[,x]	7-18
LDR	0B	Load to CR from CM augmented right half	r,¢[@][=]n[,x]	7-18
MOD	5B	Modify stack by contents of VPR halfword	r,[@]n[,x]	7-99

ASSMB CODE	MCHN CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
NOP	00	No operation	r,[@]n[,x]	7-136
OR	44	Logical OR word in CM to VPR	r,[@][=]n[,x]	7-39
OR	C4	Logical OR word in VPR to VPR	r,[@]n[,x]	7-39
OR	E4	Logical OR word in CR to VPR	r,[@]n[,x]	7-39
ORB	C7	Logical OR byte in VPR to VPR	r,n[,x]	7-42
ORB	E7	Logical OR byte in CR to VPR	r,n[,x]	7-42
ORBI	67	Logical OR immediate byte to VPR	r,i[,x]	7-43
ORH	C5	Logical OR halfword in VPR to VPR	r,n[,x]	7-39
ORH	E5	Logical OR halfword in CR to VPR	r,n[,x]	7-39
ORHI	65	Logical OR immediate halfword to VPR	r,i[,x]	7-41
ORL	45	Logical OR left half in CM to VPR	r,[@][=]n[,x]	7-44
ORR	47	Logical OR right half in CM to VPR	r,[@][=]n[,x]	7-45
POLL	F5	Poll CR and set VPR	r,n[,x]	7-133
PULL	59	Pull word from stack into VPR	r,[@]n[,x]	7-98
PUSH	58	Push word from VPR into stack	r,[@]n[,x]	7-97
RL	F2	Reset mask bits in left half of CR byte	m,n[,x]	7-107
RR	F6	Reset mask bits in right half of CR byte	m,n[,x]	7-107
SHA	60	Shift arithmetic word in VPR	r,i[,x]	7-115
SHC	6C	Shift cyclic word in VPR	r,i[,x]	7-116
SHL	64	Shift logical word in VPR	r,i[,x]	7-114
SL	FA	Set mask bits in left half of CR byte	m,n[,x]	7-106
SR	FE	Set mask bits in right half of CR byte	m,n[,x]	7-106
ST	14	Store word from VPR to CM	r,[@]n[,x]	7-2
ST	1C	Store word from VPR to CM augmented	r,¢[@]n[,x]	7-2
ST	90	Store word from VPR to VPR	r,[@]n[,x]	7-2
ST	98	Store word from VPR to CR	r,[@]n[,x]	7-2
ST	10	Store word from CR to CM	r,[@]n[,x]	7-2
ST	18	Store word from CR to CM augmented	r,¢[@]n[,x]	7-2
ST	94	Store word from CR to VPR	r,[@]n[,x]	7-2
ST	9C	Store word from CR to CR	r,[@]n[,x]	7-2
STA	1E	Store word from VPR to augmented CM absolute	r,¢[@]n[,x]	7-3
STA	16	Store word from VPR to CM absolute	r,[@]n[,x]	7-3
STB	93	Store byte from VPR to VPR	r,n[,x]	7-5

ASSMB CODE	MCHN CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
STB	9B	Store byte from VPR to CR	r, n[, x]	7-5
STB	97	Store byte from CR to VPR	r, n[, x]	7-5
STB	9F	Store byte from CR to CR	r, n[, x]	7-5
STF	1A	Store file from VPR into augmented CM	¢[@]n[, x]	7-8
STF	2A	Store file from VPR into CM	[@]n[, x]	7-8
STH	91	Store halfword from VPR to VPR	r, n[, x]	7-4
STH	99	Store halfword from VPR to CR	r, n[, x]	7-4
STH	95	Store halfword from CR to VPR	r, n[, x]	7-4
STH	9D	Store halfword from CR to CR	r, n[, x]	7-4
STL	15	Store from VPR (left or right) to left half CM	r, [@]n[, x]	7-6
STL	1D	Store from VPR to augmented left half of CM	r, ¢[@]n[, x]	7-6
STL	11	Store from CR to CM left half	r, [@]n[, x]	7-6
STL	19	Store from CR to augmented CM left half	r, ¢[@]n[, x]	7-6
STR	17	Store from VPR to CM right half	r, [@]n[, x]	7-7
STR	1F	Store from VPR to augmented CM right half	r, ¢[@]n[, x]	7-7
STR	13	Store from CR to CM right half	r, [@]n[, x]	7-7
STR	1B	Store from CR to augmented CM right half	r, ¢[@]n[, x]	7-7
SU	54	Subtract word in CM from VPR	r, [@][=]n[, x]	7-29
SU	D4	Subtract word in VPR from VPR	r, [@]n[, x]	7-29
SUB	D7	Subtract byte in VPR from VPR	r, n[, x]	7-33
SUBI	77	Subtract immediate byte from VPR	r, i[, x]	7-34
SUH	D5	Subtract halfword in VPR from VPR	r, n[, x]	7-31
SUHI	75	Subtract immediate halfword from VPR	r, i[, x]	7-32
SUI	74	Subtract immediate word from VPR	r, i[, x]	7-30
SUL	55	Subtract left half in CM from VPR	r, [@][=]n[, x]	7-35
SUR	57	Subtract right half in CM from VPR	r, [@][=]n[, x]	7-36
TAOL	EA	Test under mask fro all ones in left half of CR byte and skip if true	m, n[, x]	7-111
TAOR	EE	Test under mask for all ones in right half of CR byte and skip if true	m, n[, x]	7-111
TAZL	E2	Test under mask for all zeros in left half of CR byte and skip if true	m, n[, x]	7-112
TAZR	E6	Test under mask for all zeros in right half of CR byte and skip if true	m, n[, x]	7-112

ASSMB CODE	MCHN CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
TM	BC	Test VPR word arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
TM	AC	Test CR word arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
TMB	BF	Test VPR byte arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
TMB	AF	Test CR byte arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
TMH	BD	Test VPR halfword arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
TMH	AD	Test CR halfword arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
TN	B4	Test VPR word arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
TN	A4	Test CR word arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
TNB	B7	Test VPR byte arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
TNB	A7	Test CR byte arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
TNH	B5	Test VPR halfword arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
TNH	A5	Test CR halfword arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
TOL	CA	Test under mask for any ones in left half of CR byte and skip if true	m,n[,x]	7-109
TOR	CE	Test under mask for any ones in right half of CR byte and skip if true	m,n[,x]	7-109
TP	B8	Test VPR word arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128
TP	A8	Test CR word arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128
TPB	BB	Test VPR byte arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128
TPB	AB	Test CR byte arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128
TPH	B9	Test VPR halfword arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128
TPH	A9	Test CR halfword arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128

ASSMB CODE	MCHN CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
TROL	9A	Test under mask for any ones in left half of CR byte and reset; then skip if true	m, n[, x]	7-121
TROR	9E	Test under mask for any ones in right half of CR byte and reset; then skip if true	m, n[, x]	7-121
TRZL	92	Test under mask for any zeros in left half of CR byte and reset; then skip if true	m, n[, x]	7-120
TRZR	96	Test under mask for any zeros in right half of CR byte and reset; then skip if true	m, n[, x]	7-120
TSOL	DA	Test under mask for any ones in left half of CR byte and set; then skip if true	m, n[, x]	7-119
TSOR	DE	Test under mask for any ones in right half of CR byte and set; then skip if true	r, n[, x]	7-119
TSZL	D2	Test under mask for any zeros in left half of CR byte and set; then skip if true	m, n[, x]	7-118
TSZR	D6	Test under mask for any zeros in right half of CR byte and set; then skip if true	m, n[, x]	7-118
TZ	B0	Test VPR word arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
TZ	A0	Test CR word arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
TZB	A3	Test CR byte arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
TZB	B3	Test VPR byte arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
TZH	B1	Test VPR halfword arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
TZH	A1	Test CR halfword arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
TZL	C2	Test under mask for any zeros in left half of CR byte and skip if true	m, n[, x]	7-110
TZR	C6	Test under mask for any zeros in right half of CR byte and skip if true	m, n[, x]	7-110
VPR	82	Reset VP flag in CR	n[, x]	7-102
VPS	86	Set VP flag in CR	n[, x]	7-101
VPTO	8E	Test VP flag in CR; skip if equal to one	n[, x]	7-103
VPTZ	8A	Test VP flag in CR; skip if equal to zero	n[, x]	7-104

Table G-3. Peripheral Processor Instructions In Numeric Order By Machine Code

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
00	NOP	No operation	r,[@]n[,x]	7-136
02	BC	Branch unconditionally to CM, base relative	[@]n[,x]	7-91
04	LD	Load word to VPR from CM	r,[@][=]n[,x]	7-10
05	LDL	Load to VPR from CM left half	r,[@][=]n[,x]	7-17
06	LDA	Load word to VPR from CM absolute	r,[@][=]n[,x]	7-11
07	LDR	Load to VPR from CM right half	r,[@][=]n[,x]	7-18
08	LD	Load word to CR from augmented CM	r,¢[@][=]n[,x]	7-10
09	LDL	Load to CR from CM augmented left half	r,¢[@][=]n[,x]	7-17
0A	LDF	Load file from CM augmented into VPR	¢[@]n[,x]	7-19
0B	LDR	Load to CR from CM augmented right half	r,¢[@][=]n[,x]	7-18
0C	LD	Load word to VPR from augmented CM	r,¢[@][=]n[,x]	7-10
0D	LDL	Load to VPR from CM augmented left half	r,¢[@][=]n[,x]	7-17
0E	LDA	Load word to VPR from augmented CM	r,¢[@][=]n[,x]	7-11
0F	LDR	Load to VPR from augmented CM right half	r,¢[@][=]n[,x]	7-18
10	ST	Store word from CR to CM	r,[@]n[,x]	7-2
11	STL	Store from CR to CM left half	r,[@]n[,x]	7-6
12	BCS	Branch unconditionally to CM base relative; save PC in VPR	r,[@][=]n[,x]	7-87
13	STR	Store from CR to CM right half	r,[@]n[,x]	7-7
14	ST	Store word from VPR to CM	r,[@]n[,x]	7-2
15	STL	Store from VPR (left or right) to left half CM	r,[@]n[,x]	7-6
16	STA	Store word from VPR to CM absolute	r,[@]n[,x]	7-3
17	STR	Store from VPR to CM right half	r,[@]n[,x]	7-7
18	ST	Store word from CR to CM	r,¢[@]n[,x]	7-2
19	STL	Store from CR to augmented CM left half	r,¢[@]n[,x]	7-6
1A	STF	Store file from VPR into augmented CM	¢[@]n[,x]	7-8
1B	STR	Store from CR to augmented CM left half	r,¢[@]n[,x]	7-7
1C	ST	Store word from VPR to CM augmented	r,¢[@]n[,x]	7-2
1D	STL	Store from VPR to augmented left half CM	r,¢[@]n[,x]	7-6
1E	STA	Store word from VPR to augmented CM absolute	r,¢[@]n[,x]	7-3
1F	STR	Store from VPR to augmented CM right half	r,¢[@]n[,x]	7-7
2A	STF	Store file from VPR into CM	[@]n[,x]	7-8
30	CE	Compare word CM to VPR, skip if equal	r,[@][=]n[,x]	7-70

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
31	CEL	Compare left half CM to VPR, skip if equal	r,[@][=]n[,x]	7-76
32	BC	Branch unconditionally to augmented CM, base relative	¢[@]n[,x]	7-91
33	CER	Compare right half CM to VPR, skip if equal	r,[@][=]n[,x]	7-77
34	CN	Compare word CM to VPR, skip if not equal	r,[@][=]n[,x]	7-78
35	CNL	Compare left half CM to VPR, skip if not equal	r,[@][=]n[,x]	7-84
37	CNR	Compare right half CM to VPR, skip if not equal	r,[@][=]n[,x]	7-85
38	LD	Load word to CR from CM	r,[@][=]n[,x]	7-10
39	LDL	Load to CR from CM left half	r,[@][=]n[,x]	7-17
3A	LDF	Load file from CM into VPR	[@]n[,x]	7-19
3B	LDR	Load to CR from CM right half	r,[@][=]n[,x]	7-18
40	AN	Logical AND word in CM to VPR	r,[@][=]n[,x]	7-54
41	ANL	Logical AND left half in CM to VPR	r,[@][=]n[,x]	7-59
42	BR	Branch unconditionally to ROM	[@]n[,x]	7-93
43	ANR	Logical AND right half in CR to VPR	r,[@][=]n[,x]	7-60
44	OR	Logical OR word in CM to VPR	r,[@][=]n[,x]	7-39
45	ORL	Logical OR left half in CM to VPR	r,[@][=]n[,x]	7-44
46	BRS	Branch unconditionally to ROM, save PC in VPR	r,[@][=]n[,x]	7-94
47	ORR	Logical OR right half in CM to VPR	r,[@][=]n[,x]	7-45
48	EQ	Logical EQUIVALENCE word CM to VPR	r,[@][=]n[,x]	7-62
49	EQL	Logical EQUIVALENCE left half CM to VPR	r,[@][=]n[,x]	7-67
4A	BCA	Branch unconditionally to absolute CM	[@[=]]n[,x]	7-88
4B	EQR	Logical EQUIVALENCE right half CM to VPR	r,[@][=]n[,x]	7-68
4C	EX	Logical exclusive OR word in CM to VPR	r,[@][=]n[,x]	7-46
4D	EXL	Logical exclusive OR left half CM to VPR	r,[@][=]n[,x]	7-51
4E	BCA	Branch unconditionally to augmented absolute CM	¢[@[=]]n[,x]	7-88
4F	EXR	Logical exclusive OR right half CM to VPR	r,[@][=]n[,x]	7-52
50	AD	Add word in CM to VPR	r,[@][=]n[,x]	7-21
51	ADL	Add left half in CM to VPR	r,[@][=]n[,x]	7-27
52	BCAS	Branch unconditionally to CM absolute, save PC in VPR	r,[@][=]n[,x]	7-89

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
53	ADR	Add right half in CM to VPR	r,[@][=]n[,x]	7-28
54	SU	Subtract word in CM from VPR	r,[@][=]n[,x]	7-29
55	SUL	Subtract left half in CM from VPR	r,[@][=]n[,x]	7-35
56	BRSM	Branch unconditionally to absolute ROM, save PC in CM location, 20_{16}, augmented	n[,x]	7-95
57	SUR	Subtract right half in CM from VPR	r,[@][=]n[,x]	7-36
58	PUSH	Push word from VPR into stack	r,[@]n[,x]	7-97
59	PULL	Pull word from stack into VPR	r,[@]n[,x]	7-98
5A	BPC	Branch unconditionally to CM, relative PC	[@]n[,x]	7-90
5B	MOD	Modify stack by contents of VPR halfword	r,[@]n[,x]	7-99
5C	EXEC	Execute CM	[@[=]]n[,x]	7-134
5D	LDEA	Load effective address into VPR	r,[@][=]n[,x]	7-131
5E	BPC	Branch unconditionally to augmented CM, relative PC	¢[@]n[,x]	7-90
5F	ANAZ	Analyze CM	r,[@][=]n[,x]	7-132
60	SHA	Shift arithmetic word in VPR	r,i[,x]	7-115
61	ANHI	Logical AND immediate halfword to VPR	r,i[,x]	7-56
62	LDI	Load immediate word in CR	r,i[,x]	7-12
63	ANBI	Logical AND immediate byte to VPR	r,i[,x]	7-58
64	SHL	Shift logical word in VPR	r,i[,x]	7-114
65	ORHI	Logical OR immediate halfword to VPR	r,i[,x]	7-41
66	LDHI	Load immediate halfword in CR	r,i[,x]	7-14
67	ORBI	Logical OR immediate byte to VPR	r,i[,x]	7-43
69	EQHI	Logical EQUIVALENCE immediate halfword to VPR	r,i[,x]	7-64
6B	EQBI	Logical EQUIVALENCE immediate byte to VPR	r,i[,x]	7-66
6C	SHC	Shift cyclic word in VPR	r,i[,x]	7-116
6D	EXHI	Logical exclusive OR immediate halfword to VPR	r,i[,x]	7-48
6E	LDBI	Load immediate byte into CR	r,i[,x]	7-16
6F	EXBI	Logical exclusive OR immediate byte to VPR	r,i[,x]	7-50
70	ADI	Add immediate word to VPR	r,i[,x]	7-22
71	ADHI	Add immediate halfword to VPR	r,i[,x]	7-24
72	LDI	Load immediate word into VPR	r,i[,x]	7-12

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
73	ADBI	Add immediate byte to VPR	r,i[,x]	7-26
74	SUI	Subtract immediate word from VPR	r,i[,x]	7-30
75	SUHI	Subtract immediate halfword from VPR	r,i[,x]	7-32
76	LDHI	Load immediate halfword into VPR	r,i[,x]	7-14
77	SUBI	Subtract immediate byte from VPR	r,i[,x]	7-34
78	CEI	Compare immediate word with VPR, skip if equal	r,i[,x]	7-71
79	CEHI	Compare immediate halfword with VPR, skip if equal	r,i[,x]	7-73
7B	CEBI	Compare immediate byte with VPR, skip if equal	r,i[,x]	7-75
7C	CNI	Compare immediate word with VPR, skip if not equal	r,i[,x]	7-79
7D	CNHI	Compare immediate halfword with VPR, skip if not equal	r,i[,x]	7-81
7E	LDBI	Load immediate byte into VPR	r,i[,x]	7-16
7F	CNBI	Compare immediate byte with VPR, skip if not equal	r,i[,x]	7-83
80	LD	Load word to VPR from VPR	r,[@]n[,x]	7-10
81	LDH	Load halfword to VPR from VPR	r,n[,x]	7-13
82	VPR	Reset VP flag in CR	n[,x]	7-102
83	LDB	Load byte to VPR from VPR	r,n[,x]	7-15
84	LD	Load word to CR from VPR	r,[@]n[,x]	7-10
85	LDH	Load halfword to CR from VPR	r,n[,x]	7-13
86	VPS	Set VP flag in CR	n[,x]	7-101
87	LDB	Load byte to CR from VPR	r,n[,x]	7-15
88	LD	Load word to VPR from CR	r,[@]n[,x]	7-10
89	LDH	Load halfword to VPR from CR	r,n[,x]	7-13
8A	VPTZ	Test VP flag in CR, skip if equal to zero	n[,x]	7-104
8B	LDB	Load byte to VPR from CR	r,n[,x]	7-15
8C	LD	Load word to CR from CR	r,[@]n[,x]	7-10
8D	LDH	Load halfword to CR from CR	r,n[,x]	7-13
8E	VPTO	Test VP flag in CR, skip if equal to one	n[,x]	7-103
8F	LDB	Load byte to CR from CR	r,n[,x]	7-15
90	ST	Store word from VPR to VPR	r,[@]n[,x]	7-2

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
91	STH	Store halfword from VPR to VPR	r, n[, x]	7-4
92	TRZL	Test mask for any zeros in left half of CR byte, skip if true	m, n[, x]	7-120
93	STB	Store byte from VPR to VPR	r, n[, x]	7-5
94	ST	Store word from CR to VPR	r, [@]n[, x]	7-2
95	STH	Store halfword from CR to VPR	r, n[, x]	7-4
96	TRZR	Test under mask for any zeros in right half of CR byte, skip if true	m, n[, x]	7-120
97	STB	Store byte from CR to VPR	r, n[, x]	7-5
98	ST	Store word from VPR to CR	r, [@]n[, x]	7-2
99	STH	Store halfword from VPR to CR	r, n[, x]	7-4
9A	TROL	Test under mask for any ones in left half of CR byte, skip if true	m, n[, x]	7-121
9B	STB	Store byte from VPR to CR	r, n[, x]	7-5
9C	ST	Store word from CR to CR	r, [@]n[, x]	7-2
9D	STH	Store halfword from CR to CR	r, n[, x]	7-4
9E	TROR	Test under mask for any ones in right half of CR byte, skip if true	m, n[, x]	7-121
9F	STB	Store byte from CR to CR	r, n[, x]	7-5
A0	TZ	Test CR word arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
A1	TZH	Test CR halfword arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
A3	TZB	Test CR byte arithmetically; branch if equal to zero	r, [@][=]n[, x]	7-126
A4	TN	Test CR word arithmetically; branch if not equal to zero	r, [@][=]n[, x]	7-127
A5	TNH	Test CR halfword arithmetically; branch if not equal to zero	r, [@][=]n[, x]	7-127
A7	TNB	Test CR byte arithmetically; branch if not equal to zero	r, [@][=]n[, x]	7-127
A8	TP	Test CR word arithmetically; branch if greater than or equal to zero	r, [@][=]n[, x]	7-128
A9	TPH	Test CR halfword arithmetically; branch if greater than or equal to zero	r, [@][=]n[, x]	7-128
AB	TPB	Test CR byte arithmetically; branch if greater than or equal to zero	r, [@][=]n[, x]	7-128

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
AC	TM	Test CR word arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
AD	TMH	Test CR halfword arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
AE	BPCS	Branch unconditionally to CM, relative PC, save PC in VPR	r,[@][=]n[,x]	7-92
AF	TMB	Test CR byte arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
B0	TZ	Test VPR word arithmetically; branch if equal to zero	r,[@][=]n[,x]	7-126
B1	TZH	Test VPR halfword arithmetically; branch if equal to zero	r,[@][=]n[x]	7-126
B2	IBZ	Increment VPR by 1; branch if result equals zero	r,[@][=]n[,x]	7-123
B3	TZB	Test VPR byte arithmetically; branch if equal to zero	r,[@][=]n[,x]	7-126
B4	TN	Test VPR word arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
B5	TNH	Test VPR halfword arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
B6	IBN	Increment VPR by 1; branch if result not equal to zero	r,[@][=]n[,x]	7-123
B7	TNB	Test VPR byte arithmetically; branch if not equal to zero	r,[@][=]n[,x]	7-127
B8	TP	Test VPR word arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128
B9	TPH	Test VPR halfword arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128
BA	DBZ	Decrement VPR by 1; branch if result equals zero	r,[@][=]n[,x]	7-124
BB	TPB	Test VPR byte arithmetically; branch if greater than or equal to zero	r,[@][=]n[,x]	7-128
BC	TM	Test VPR word arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
BD	TMH	Test VPR halfword arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129
BE	DBN	Decrement VPR by 1; branch if result is not equal to zero	r,[@][=]n[,x]	7-124
BF	TMB	Test VPR byte arithmetically; branch if less than zero	r,[@][=]n[,x]	7-129

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
C0	AN	Logical AND word in VPR to VPR	r,[@]n[,x]	7-54
C1	ANH	Logical AND halfword in VPR to VPR	r,n[,x]	7-55
C2	TZL	Test under mask for any zeros in left half of CR byte and skip if true	m,n[,x]	7-110
C3	ANB	Logical AND byte in VPR to VPR	r,n[,x]	7-57
C4	OR	Logical OR word in VPR to VPR	r,[@]n[,x]	7-39
C5	ORH	Logical OR halfword in VPR to VPR	r,n[,x]	7-39
C6	TZR	Test under mask for any zeros in right half of CR byte and skip if true	m,n[,x]	7-110
C7	ORB	Logical OR byte in VPR to VPR	r,n[,x]	7-42
C8	EQ	Logical EQUIVALENCE word in VPR to VPR	r,[@]n[,x]	7-60
C9	EQH	Logical EQUIVALENCE halfword VPR to VPR	r,n[,x]	7-63
CA	TOL	Test under mask for any ones in left half of CR byte; skip if true	m,n[,x]	7-109
CB	EQB	Logical EQUIVALENCE byte VPR to VPR	r,n[,x]	7-65
CC	EX	Logical exclusive OR word in VPR to VPR	r,[@]n[,x]	7-46
CD	EXH	Logical exclusive OR halfword in VPR to VPR	r,n[,x]	7-47
CE	TOR	Test under mask for any ones in right half of CR byte and skip if true	m,n[,x]	7-109
CF	EXB	Logical exclusive OR byte in VPR to VPR	r,n[,x]	7-49
D0	AD	Add word in VPR to VPR	r,[@]n[,x]	7-21
D1	ADH	Add halfword in VPR to VPR	r,n[,x]	7-23
D2	TSZL	Test under mask for any zeros in left half of CR byte and skip if true	m,n[,x]	7-118
D3	ADB	Add byte in VPR to VPR	r,n[,x]	7-25
D4	SU	Subtract word in VPR from VPR	r,[@]n[,x]	7-29
D5	SUH	Subtract halfword in VPR from VPR	r,n[,x]	7-31
D6	TSZR	Test under mask for any zeros in right half of CR byte and skip if true	m,n[,x]	7-118
D7	SUB	Subtract byte in VPR from YPR	r,n[,x]	7-33
D8	CE	Compare word VPR to VPR, skip if equal	r,[@]n[,x]	7-70
D9	CEH	Compare halfword VPR to VPR, skip if equal	r,n[,x]	7-72
DA	TSOL	Test under mask for any ones in left half of CR byte and skip if true	m,n[,x]	7-119
DB	CEB	Compare byte VPR to VPR, skip if equal	r,n[,x]	7-73
DC	CN	Compare word VPR to VPR, skip if not equal	r,[@]n[,x]	7-78

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
DD	CNH	Compare halfword VPR to VPR, skip if not equal	r, n[, x]	7-80
DE	TSOR	Test under mask for any ones in right half of CR byte and skip if true	m, n[, x]	7-119
DF	CNB	Compare byte VPR to VPR, skip if not equal	r, n[, x]	7-82
E0	AN	Logical AND word in CR to VPR	r, [@]n[, x]	7-54
E1	ANH	Logical AND halfword in CR to VPR	r, n[, x]	7-55
E2	TAZL	Test under mask for all zeros in left half of CR byte and skip if true	m, n[, x]	7-112
E3	ANB	Logical AND byte in CR to VPR	r, n[, x]	7-57
E4	OR	Logical OR word in CR to VPR	r, [@]n[, x]	7-39
E5	ORH	Logical OR halfword in CR to VPR	r, n[, x]	7-39
E6	TAZR	Test under mask for all zeros in right half of CR byte and skip if true	m, n[, x]	7-112
E7	ORB	Logical OR byte in CR to VPR	r, n[, x]	7-42
E8	EQ	Logical EQUIVALENCE word CR to VPR	r, [@]n[, x]	7-62
E9	EQH	Logical EQUIVALENCE halfword CR to VPR	r, n[, x]	7-63
EA	TADL	Test under mask for all ones in left half of CR byte and skip if true	m, n[, x]	7-111
EB	EQB	Logical EQUIVALENCE byte CR to VPR	r, n[, x]	7-65
EC	EX	Logical exclusive OR word in CR to VPR	r, [@]n[, x]	7-46
ED	EXH	Logical exlusive OR halfword in CR to VPR	r, n[, x]	7-47
EE	TAOR	Test under mask for all ones in right half CR byte and skip if true	m, n[, x]	7-111
EF	EXB	Logical exclusive OR byte in CR to VPR	r, n[, x]	7-48
F2	RL	Reset mask bits in left half of CR byte	m, n[, x]	7-107
F4	LDMB	Load VP base from VPR to CR	[@]n[, x]	7-135
F5	POLL	Poll CR and set VPR	r, n[, x]	7-133
F6	RR	Reset mask bits in right half of CR byte	m, n[, x]	7-107
F8	CE	Compare word CR to VPR, skip if equal	r, [@]n[, x]	7-70
F9	CEH	Compare halfword CR to VPR, skip if equal	r, n[, x]	7-72
FA	SL	Set mask bits in left half of CR byte	m, n[, x]	7-106
FB	CEB	Compare byte VPR to VPR, skip if equal	r, n[, x]	7-74
FC	CN	Compare word CR to VPR, skip if not equal	r, [@]n[, x]	7-78
FD	CNH	Compare halfword CR to VPR, skip if not equal	r, n[, x]	7-80

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
FE	SR	Set mask bits in right half of CR byte	m, n[, x]	7-106
FF	CNB	Compare byte CR to VPR, skip if not equal	r, n[, x]	7-82

APPENDIX H: CENTRAL PROCESSOR MACHINE INSTRUCTIONS

Table H-1. Central Processor Scalar Instructions By Logical Grouping

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
ST	Store arithmetic register, singleword	24	r,[@]n[,x]	7-26
ST	Store base register, singleword	28	r,[@]n[,x]	7-26
ST	Store index register or vector parameter register, singleword	2C	r,[@]n[,x]	7-26
STH	Store halfword, arithmetic register	25	r,[@]n[,x]	7-27
STR	Store arithmetic register right halfword into memory right halfword, indexed	2D	r,[@]n[,x]	7-28
STL	Store arithmetic register left halfword into memory right halfword, indexed	29	r,[@]n[,x]	7-29
STD	Store arithmetic register, doubleword	27	r,[@]n[,x]	7-30
STZ	Store zero, word	20	[@]n[,x]	7-31
STZH	Store zero, halfword	21	[@]n[,x]	7-32
STZD	Store zero, doubleword	23	[@]n[,x]	7-33
STN	Store negative, fixed point word	34	r,[@]n[,x]	7-34
STNH	Store negative, fixed point halfword	35	r,[@]n[,x]	7-35
STNF	Store negative, floating point word	36	r,[@]n[,x]	7-36
STND	Store negative, floating point doubleword	37	r,[@]n[,x]	7-37
STO	Store ones complement, word	2E	r,[@]n[,x]	7-38
STOH	Store ones complement, halfword	2A	r,[@]n[,x]	7-39
STF	Store base register file A, M=0	2B	m,[@]n[,x]	7-40
STF	Store base register file B, M=1	2B	m,[@]n[,x]	7-40
STF	Store arithmetic register file C, M=2	2B	m,[@]n,[,x]	7-40
STF	Store arithmetic register file D, M=3	2B	m,[@]n[,x]	7-40
STF	Store index register file X, M=4	2B	m,[@]n[,x]	7-40
STF	Store vector parameter register file V, M=5	2B	m,[@]n[,x]	7-40
STFM	Store all six eight-word register files	2F	[@]n[,x]	7-41
L	Load arithmetic register, singleword	14	r,[@][=]n[,x]	7-3
L	Load base register, singleword	18	r,[@][=]n[,x]	7-3
L	Load index register or vector parameter register, singleword	1C	r,[@][=]n[,x]	7-3
LH	Load arithmetic register, halfword	15	r,[@][=]n[,x]	7-4
LR	Load memory right halfword, indexed, into arithmetic register right halfword	1D	r,[@][=]n[,x]	7-5
LL	Load memory right halfword, indexed, into arithmetic register left halfword	19	r,[@][=]n[,x]	7-6

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
LD	Load arithmetic register, doubleword	17	r,[@][=]n[,x]	7-7
LI	Load immediate into arithmetic register singleword	54	r,i[,x]	7-8
LI	Load immediate into index register, or vector parameter register, singleword	5C	r,i[,x]	7-8
LIH	Load immediate into arithmetic register, halfword	55	r,i[,x]	7-9
LN	Load negative, fixed point singleword, arithmetic register	30	r,[@][=]n[,x]	7-10
LNH	Load negative, fixed point halfword, arithmetic register	31	r,[@][=]n[,x]	7-11
LNF	Load negative, floating point singleword, arithmetic register	32	r,[@][=]n[,x]	7-12
LND	Load negative, floating point doubleword, arithmetic register	33	r,[@][=]n[,x]	7-13
LM	Load magnitude, fixed point singleword, arithmetic register	3C	r,[@][=]n[,x]	7-14
LMH	Load magnitude, fixed point halfword, arithmetic register	3D	r,[@][=]n[,x]	7-15
LMF	Load magnitude, floating point singleword, arithmetic register	3E	r,[@][=]n[,x]	7-16
LMD	Load magnitude, floating point doubleword, arithmetic register	3F	r,[@][=]n[,x]	7-17
LNM	Load negative magnitude, fixed point singleword, arithmetic register	38	r,[@][=]n[,x]	7-18
LNMH	Load negative magnitude, fixed point halfword, arithmetic register	39	r,[@][=]n[,x]	7-19
LNMF	Load negative magnitude, floating point singleword, arithmetic register	3A	r,[@][=]n[,x]	7-20
LNMD	Load negative magnitude, floating point doubleword, arithmetic	3B	r,[@][=]n[,x]	7-21
LO	Load arithmetic register with ones complement, singleword	1E	r,[@][=]n[,x]	7-22
LF	Load base register file A, M=0	1B	m,[@]n[,x]	7-23
LF	Load base register file B, M=1	1B	m,[@]n[,x]	7-23
LF	Load arithmetic register file C, M=2	1B	m,[@]n[,x]	7-23
LF	Load arithmetic register file D, M=3	1B	m,[@]n[,x]	7-23
LF	Load index register file X, M=4	1B	m,[@]n[,x]	7-23

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
LF	Load vector parameter register file V, M=5	1B	m,[@]n[,x]	7-23
LFM	Load all six eight-word register files	1F	[@]n[,x]	7-24
A	Add to arithmetic register, fixed point single-word	40	r,[@][=]n[,x]	7-43
A	Add to base register, fixed point singleword	60	r,[@][=]n[,x]	7-43
A	Add to index or vector parameter register, fixed point singleword	62	r,[@][=]n[,x]	7-43
AH	Add to arithmetic register, fixed point halfword	41	r,[@][=]n[,x]	7-44
AF	Add to arithmetic register, floating point single-word	42	r,[@][=]n[,x]	7-45
AFD	Add to arithmetic register, floating point doubleword	43	r,[@][=]n[,x]	7-46
AI	Add immediate to arithmetic register, fixed point singleword	50	r,i[,x]	7-47
AI	Add immediate to base register, fixed point singleword	70	r,i[,x]	7-47
AI	Add immediate to index or vector parameter register, fixed point singleword	72	r,i[,x]	7-47
AIH	Add immediate to arithmetic register, fixed point halfword	51	r,i[,x]	7-48
AM	Add magnitude to arithmetic register, fixed point singleword	44	r,[@][=]n[,x]	7-49
AMH	Add magnitude to arithmetic register, fixed point halfword	45	r,[@][=]n[,x]	7-50
AMF	Add magnitude to arithmetic register, floating point singleword	46	r,[@][=]n[,x]	7-51
AMFD	Add magnitude to arithmetic register, floating point doubleword	47	r,[@][=]n[,x]	7-52
S	Subtract from arithmetic register, fixed point singleword	48	r,[@][=]n[,x]	7-53
SH	Subtract from arithmetic register, fixed point halfword	49	r,[@][=]n[,x]	7-54
SF	Subtract from arithmetic register, floating point singleword	4A	r,[@][=]n[,x]	7-55
SFD	Subtract from arithmetic register, floating point doubleword	4B	r,[@][=]n[,x]	7-56
SI	Subtract immediate from arithmetic register, fixed point singleword	58	r,i[,x]	7-57

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
SIH	Subtract immediate from arithmetic register, fixed point halfword	59	r,i[,x]	7-57
SM	Subtract magnitude from arithmetic register, fixed point singleword	4C	r,[@][=]n[,x]	7-59
SMH	Subtract magnitude from arithmetic register, fixed point halfword	4D	r,[@][=]n[,x]	7-60
SMF	Subtract magnitude from arithmetic register, floating point singleword	4F	r,[@][=]n[,x]	7-61
SMFD	Subtract magnitude from arithmetic register, floating point doubleword	4F	r,[@][=]n[,x]	7-62
M	Multiply, fixed point singleword - arithmetic register	6C	r,[@][=]n[,x]	7-63
M	Multiply, fixed point singleword - base register	68	r,[@][=]n[,x]	7-63
M	Multiply, fixed point singleword - index or vector parameter register	6A	r,[@][=]n[,x]	7-63
MH	Multiply, fixed point halfword - arithmetic register	6D	r,[@][=]n[,x]	7-64
MF	Multiply, floating point singleword - arithmetic register	6E	r,[@][=]n[,x]	7-65
MFD	Multiply, floating point doubleword - arithmetic register	6F	r,[@][=]n[,x]	7-66
MI	Multiply immediate, fixed point singleword - arithmetic register	7C	r,i[,x]	7-67
MI	Multiply immediate, fixed point singleword - base register	78	r,i[,x]	7-67
MI	Multiply immediate, fixed point singleword- index or vector parameter register	7A	r,i[,x]	7-67
MIH	Multiply immediate, fixed point halfword - arithmetic register	7D	r,i[,x]	7-68
D	Divide into arithmetic register, fixed point singleword	64	r,[@][=]n[,x]	7-69
DH	Divide into arithmetic register, fixed point halfword	65	r,[@][=]n[,x]	7-70
DF	Divide into arithmetic register, floating point singleword	66	r,[@][=]n[,x]	7-71
DFD	Divide into arithmetic register, floating point doubleword	67	r,[@][=]n[,x]	7-72
DI	Divide immediate into arithmetic register, fixed point singleword	74	r,i[,x]	7-73

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
DIH	Divide immediate into arithmetic register, fixed point halfword	75	r,i[,x]	7-74
AND	AND, singleword - arithmetic register	E0	r,[@][=]n[,x]	7-76
ANDD	AND, doubleword - arithmetic register	E1	r,[@][=]n[,x]	7-77
ANDI	AND immediate, singleword - arithmetic register	F0	r,i[,x]	7-78
OR	OR, singleword - arithmetic register	E4	r,[@][=]n[,x]	7-79
ORD	OR, doubleword - arithmetic register	E5	r,[@][=]n[,x]	7-80
ORI	OR immediate, singleword - arithmetic register	F4	r,i[,x]	7-81
XOR	Exclusive OR, singleword - arithmetic register	E8	r,[@][=]n[,x]	7-82
XORD	Exclusive OR, doubleword - arithmetic register	E9	r,[@][=]n[,x]	7-83
XORI	Exclusive OR immediate, singleword-arithmetic register	F8	r,i[,x]	7-84
EQC	Equivalence, singleword - arithmetic register	EC	r,[@][=]n[,x]	7-85
EQCD	Equivalence, doubleword - arithmetic register	ED	r,[@][=]n[,x]	7-86
EQCI	Equivalence immediate, singleword - arithmetic register	FC	r,i[,x]	7-87
SA	Arithmetic shift, fixed point singleword - arithmetic register	C0	r,i[,x]	7-93
SAH	Arithmetic shift, fixed point halfword - arithmetic register	C1	r,i[,x]	7-94
SAD	Arithmetic shift, fixed point doubleword - arithmetic register	C3	r,i[,x]	7-95
SL	Logical shift, singleword - arithmetic register	C4	r,i[,x]	7-96
SLH	Logical shift, halfword - arithmetic register	C5	r,i[,x]	7-97
SLD	Logical shift, doubleword - arithmetic register	C7	r,i[,x]	7-98
SC	Circular shift, singleword - arithmetic register	CC	r,i[,x]	7-99
SCH	Circular shift, halfword - arithmetic register	CD	r,i[,x]	7-100
SCD	Circular shift, doubleword - arithmetic register	CF	r,i[,x]	7-101
RVS	Bit reversal, singleword - arithmetic register	C6	r,i[,x]	7-102
C	Compare arithmetic register, fixed point singleword	C8	r,[@][=]n[,x]	7-104
C	Compare index or vector register, fixed point singleword	CE	r,[@][=]n[,x]	7-104
CH	Compare arithmetic register, fixed point halfword	C9	r,[@][=]n[,x]	7-105

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
CF	Compare arithmetic register, floating point singleword	CA	r,[@][=]n[,x]	7-106
CFD	Compare arithmetic register, floating point doubleword	CB	r,[@][=]n[,x]	7-107
CI	Compare immediate arithmetic register, fixed point singleword	D8	r,i[,x]	7-108
CI	Compare index or vector register with immediate, singleword	DE	r,i[,x]	7-108
CIH	Compare arithmetic register immediate, fixed point halfword	D9	r,i[,x]	7-109
CAND	Compare logical AND, singleword - arithmetic register	E2	r,[@][=]n[,x]	7-110
CANDD	Compare logical AND, doubleword - arithmetic register	E3	r,[@][=]n[,x]	7-111
CANDI	Compare immediate logical AND, singleword - arithmetic register	F2	r,i[,x]	7-112
COR	Compare logical OR, singleword - arithmetic register	E6	r,[@][=]n[,x]	7-113
CORD	Compare logical OR, doubleword - arithmetic register	E7	r,[@][=]n[,x]	7-114
CORI	Compare immediate logical OR, singleword - arithmetic register	F6	r,i[,x]	7-115
ISE	Increment arithmetic register, test, and skip on equal	80	r,[@][=]n[,x]	7-117
ISNE	Increment arithmetic register, test, and skip on not equal	81	r,[@][=]n[,x]	7-118
DSE	Decrement arithmetic register, test, and skip on equal	82	r,[@][=]n[,x]	7-119
DSNE	Decrement arithmetic register, test, and skip on not equal	83	r,[@][=]n[,x]	7-120
BCC	Branch on compare code true	91	m,[@[=]]n[,x]	7-132
NOP	Take next instruction, Assembler supplies R field of zero	91	[@[=]]n[,x]	7-132
BE	Branch on compare code of equal, Assembler supplies R field of one	91	[@[=]]n[,x]	7-132
BG	Branch on compare code of greater than, Assembler supplies R field of 2	91	[@[=]]n[,x]	7-132
BGE	Branch on compare code of greater than or equal, Assembler supplies R field of 3	91	[@[=]]n[,x]	7-132

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
BL	Branch on compare code of less than, Assembler supplies R field of 4	91	[@[=]]n[,x]	7-132
BLE	Branch on compare code of less than or equal, Assembler supplies R field of 5	91	[@[=]]n[,x]	7-132
BNE	Branch on compare code of not equal, Assembler supplies R field of 6	91	[@[=]]n[,x]	7-132
B	Unconditional branch, Assembler supplies R field of 7	91	[@[=]]n[,x]	7-132
BCZ	Branch on compare code of all bits are zero, Assembler supplies R field of one	91	[@[=]]n[,x]	7-132
BCO	Branch on compare code of all bits are one, Assembler supplies R field of 2	91	[@[=]]n[,x]	7-132
BCNM	Branch on compare code of not mixed, Assembler supplies R field of 3	91	[@[=]]n[,x]	7-132
BCM	Branch on compare code of mixed zeros and ones, Assembler supplies R field of 4	91	[@[=]]n[,x]	7-132
BCNO	Branch on compare code of not all ones, Assembler supplies R field of 5	91	[@[=]]n[,x]	7-132
BCNZ	Branch on compare code of not all zeros, Assembler supplies the R field of 6	91	[@[=]]n[,x]	7-132
BRC	Branch on result code true	95	m,[@[=]]n[,x]	7-133
BZ	Branch on result code of zero, Assembler supplies the R field of one	95	[@[=]]n[,x]	7-133
BPL	Branch on result code of positive, Assembler supplies the R field of 2	95	[@[=]]n[,x]	7-133
BZP	Branch on result code of zero or positive, Assembler supplies the R field of 3	95	[@[=]]n[,x]	7-133
BMI	Branch on result code of negative, Assembler supplies the R field of 4	95	[@[=]]n[,x]	7-133
BZM	Branch on result code of zero or negative, Assembler supplies the R field of 5	95	[@[=]]n[,x]	7-133
BNZ	Branch on result code of not zero, Assembler supplies the R field of 6	95	[@[=]]n[,x]	7-133
BRZ	Branch on result code of all bits are zero, Assembler supplies the R field of one	95	[@[=]]n[,x]	7-133
BRO	Branch on result code of all bits are one, Assembler supplies the R field of 2	95	[@[=]]n[,x]	7-133
BRNM	Branch on result code of bits not mixed zeros and ones, Assembler supplies the R field of 3	95	[@[=]]n[,x]	7-133

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
BRM	Branch on result code of bits mixed zeros and ones, Assembler supplies the R field of 4	95	[@[=]]n[,x]	7-133
BRNO	Branch on result code of not all bits ones, Assembler supplies the R field of 5	95	[@[=]]n[,x]	7-133
BRNZ	Branch on result code of not all bits zeros, Assembler supplies the R field of 6	95	[@[=]]n[,x]	7-133
BAE	Branch on arithmetic exception condition true	9D	m,[@[=]]n[,x]	7-134
BU	Branch on floating point exponent under flow, Assembler supplies R field of one	9D	[@[=]]n[,x]	7-134
BO	Branch on floating point exponent overflow, Assembler supplies R field of 2	9D	[@[=]]n[,x]	7-134
BUO	Branch on floating point exponent underflow or overflow, Assembler supplies R field of 3	9D	[@[=]]n[,x]	7-134
BX	Branch on fixed point overflow, Assembler supplies R field of 4	9D	[@[=]]n[,x]	7-134
BXU	Branch on fixed point overflow or floating point exponent underflow, Assembler supplies R field of 5	9D	[@[=]]n[,x]	7-134
BXO	Branch on fixed point overflow of floating point exponent overflow, Assembler supplies R field of 6	9D	[@[=]]n[,x]	7-134
BXUO	Branch on fixed point overflow or floating point exponent overflow or underflow, Assembler supplies R field of 7	9D	[@[=]]n[,x]	7-134
BD	Branch on divide check, Assembler supplies R field of 8	9D	[@[=]]n[,x]	7-134
BDU	Branch on divide check or floating point exponent underflow, Assembler supplies R field of 9	9D	[@[=]]n[,x]	7-134
BDO	Branch on divide check on floating point exponent overflow, Assembler supplies R field of A	9D	[@[=]]n[,x]	7-134
BDUO	Branch on divide check or floating point exponent overflow or underflow, Assembler supplies R field of B	9D	[@[=]]n[,x]	7-134
BDX	Branch on divide check or fixed point overflow, Assembler supplies R field of C	9D	[@[=]]n[,x]	7-134
BDXU	Branch on divide check or fixed point overflow or floating point exponent underflow, Assembler supplies R field of D	9D	[@[=]]n[,x]	7-134

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
BDXO	Branch on divide check or fixed point overflow or floating point exponent overflow, Assembler supplies R field of E	9D	[@[=]]n[,x]	7-134
BDXUO	Branch on divide check or fixed point overflow or floating point exponent overflow or underflow, Assembler supplies R field of F	9D	[@[=]]n[,x]	7-134
BXEC	Branch on Execute branch condition true, Assembler supplies R field of one or odd	9C	[@]n[,x]	7-135
IBZ	Increment arithmetic register, test, and branch on zero	88	r,[@[=]]n[,x]	7-122
IBZ	Increment index or vector register, test, and branch on zero	8C	r,[@[=]]n[,x]	7-122
IBNZ	Increment arithmetic register, test, and branch on not zero	89	r,[@[=]]n[,x]	7-123
IBNZ	Increment index or vector register, and branch on not zero	8D	r,[@[=]]n[,x]	7-123
DBZ	Decrement arithmetic register, test, and branch zero	8A	r,[@[=]]n[,x]	7-124
DBZ	Decrement index or vector register, test, and branch on zero	8E	r,[@[=]]n[,x]	7-124
DBNZ	Decrement arithmetic register, test, and branch on not zero	8B	r,[@[=]]n[,x]	7-124
DBNZ	Decrement index or vector register, test, and branch on not zero	8F	r,[@[=]]n[,x]	7-124
BCLE	Branch on arithmetic register less than or equal	84	r,r,n	7-128
BCLE	Branch on index or vector register less than or equal	86	r,r,n	7-128
BCG	Branch on arithmetic register greater than	85	r,r,n	7-129
BCG	Branch on index or vector register greater than	87	r,r,n	7-129
BLB	Branch and load base register with program counter	98	r,[@[=]]n[,x]	7-137
BLX	Branch and load index or vector register with program counter	99	r,[@[=]]n[,x]	7-138
PSH	Push word into last-in-first-out stack	93	r,[@]n[,x]	7-141
PUL	Pull word from last-in-first-out stack	97	r,[@]n[,x]	7-142
MOD	Modify stack parameter doubleword	9F	r,[@]n[,x]	7-143
FLFX	Convert floating point singleword to fixed point singleword	A0	r,[@]n[,x]	7-148

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
FLFH	Convert floating point singleword to fixed point halfword	A1	r,[@]n[,x]	7-149
FDFX	Convert floating point doubleword to fixed point singleword	A2	r,[@]n[,x]	7-150
FXFL	Convert fixed point singleword to floating point singleword	A8	r,[@]n[,x]	7-151
FHFL	Convert fixed point halfword to floating point singleword	A9	r,[@]n[,x]	7-152
FXFD	Convert fixed point singleword to floating point doubleword	AA	r,[@]n[,x]	7-153
FHFD	Convert fixed point halfword to floating point doubleword	AB	r,[@]n[,x]	7-154
NFX	Normalize fixed point singleword	AC	r,[@]n[,x]	7-155
NFH	Normalize fixed point halfword	AD	r,[@]n[,x]	7-156
XCH	Exchange - arithmetic register with effective address	1A	r,[@]n[,x]	7-158
LLA	Load look ahead	16	i	7-159
LEA	Load effective address into base register	52	r,[@][=]n[,x]	7-160
LEA	Load effective address into index or vector register	56	r,[@][=]n[,x]	7-160
XEC	Execute addressed instruction in line	96	[@[=]]n[,x]	7-161
INT	Interpret - arithmetic register	92	r,[@][=]n[,x]	7-162
MCP	Monitor call and proceed	90	i[,x]	7-163
MCW	Monitor call and wait	94	i[,x]	7-164
LAM	Load arithmetic mask	12	[@[=]]n[,x]	7-166
LAC	Load arithmetic exception condition	13	[@[=]]n[,x]	7-167
SPS	Store program status word	22	[@]n[,x]	7-168
VECTL	Load and execute vector parameter file, Assembler supplies R field of zero	B0	[@]n[,x]	8-4
VECT	Execute vector parameter file, Assembler supplies R field of one	B0	[@]n[,x]	8-5

Table H-2. Central Processor Scalar Instructions In Alphabetical Order By Assembler Code

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
A	Add to arithmetic register, fixed point single-word	40	r,[@][=]n[,x]	7-43
A	Add to base register, fixed point singleword	60	r,[@][=]n[,x]	7-43
A	Add to index or vector parameter register,	62	r,[@][=]n[,x]	7-43
AF	Add to arithmetic register, floating point singleword	42	r,[@][=]n[,x]	7-45
AFD	Add to arithmetic register, floating point doubleword	43	r,[@][=]n[,x]	7-46
AH	Add to arithmetic register, fixed point halfword	41	r,[@][=]n[,x]	7-44
AI	Add immediate to arithmetic register, fixed point singleword	50	r,i[,x]	7-47
AI	Add immediate to base register, fixed point singleword	70	r,i[,x]	7-47
AI	Add immediate to index or vector parameter register, fixed point singleword	72	r,i[,x]	7-47
AIH	Add immediate to arithmetic register, fixed point halfword	51	r,i[,x]	7-48
AM	Add magnitude to arithmetic register, fixed point singleword	44	r,[@][=]n[,x]	7-49
AMF	Add magnitude to arithmetic register, floating point singleword	46	r,[@][=]n[,x]	7-51
AMFD	Add magnitude to arithmetic register, floating point doubleword	47	r,[@][=]n[,x]	7-52
AMH	Add magnitude to arithmetic register, fixed point halfword	45	r,[@][=]n[,x]	7-50
AND	AND, singleword - arithmetic register	E0	r,[@][=]n[,x]	7-76
ANDD	AND, doubleword - arithmetic register	E1	r,[@][=]n[,x]	7-77
ANDI	AND immediate, singleword - arithmetic register	F0	r,i[,x]	7-78
B	Unconditional branch, Assembler supplies R field of 7	91	[@[=]]n[,x]	7-132
BAE	Branch on arithmetic exception condition true	9D	m,[@[=]]n[,x]	7-134
BCC	Branch on compare code true	91	m,[@[=]]n[,x]	7-132
BCG	Branch on arithmetic register greater than	85	r,r,n	7-129
BCG	Branch on index or vector register greater than	87	r,r,n	7-129
BCLE	Branch on arithmetic register less than or equal	84	r,r,n	7-128

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
BCLE	Branch on index or vector register less than or equal	86	r, r, n	7-128
BCM	Branch on compare code of mixed zeros and ones, Assembler supplies R field of 4	91	[@[=]]n[,x]	7-132
BCNM	Branch on compare code of not mixed, Assembler supplies R field of 3	91	[@[=]]n[,x]	7-132
BCNO	Branch on compare code of not all ones, Assembler supplies R field of 5	91	[@[=]]n[,x]	7-132
BCNZ	Branch on compare code of not all zeros, Assembler supplies the R field of 6	91	[@[=]]n[,x]	7-132
BCO	Branch on compare code of all bits are one, Assembler supplies R field of 2	91	[@[=]]n[,x]	7-132
BCZ	Branch on compare code of all bits are zero, Assembler supplies R field of one	91	[@[=]]n[,x]	7-132
BD	Branch on divide check, Assembler supplies R field of 8	9D	[@[=]]n[,x]	7-134
BDO	Branch on divide check on floating point exponent overflow, Assembler supplies R field of A	9D	[@[=]]n[,x]	7-134
BDU	Branch on divide check or floating point exponent underflow, Assembler supplies R field of 9	9D	[@[=]]n[,x]	7-134
BDUO	Branch on divide check or floating point exponent overflow or underflow, Assembler supplies R field of B	9D	[@[=]]n[,x]	7-134
BDX	Branch on divide check or fixed point overflow, Assembler supplies R field of C	9D	[@[=]]n[,x]	7-134
BDXO	Branch on divide check or fixed point overflow or floating point exponent overflow, Assembler supplies R field of E	9D	[@[=]]n[,x]	7-134
BDXU	Branch on divide check or fixed point overflow or floating point exponent underflow, Assembler supplies R field of D	9D	[@[=]]n[,x]	7-134
BDXUO	Branch on divide check or fixed point overflow or floating point exponent overflow or underflow, Assembler supplies R field of F	9D	[@[=]]n[,x]	7-134
BE	Branch on compare code of equal, Assembler supplies R field of one	91	[@[=]]n[,x]	7-132
BG	Branch on compare code of greater than, Assembler supplies R field of 2	91	[@[=]]n[,x]	7-132

ASSME CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
BGE	Branch on compare code of greater than or equal, Assembler supplies R field of 3	91	[@[=]]n[,x]	7-132
BL	Branch on compare code of less than, Assembler supplies R field of 4	91	[@[=]]n[,x]	7-132
BLB	Branch and load base register with program counter	98	r,[@[=]]n[,x]	7-137
BLE	Branch on compare code of less than or equal, Assembler supplies R field of 5	91	[@[=]]n[,x]	7-132
BLX	Branch and load index or vector register with program counter	99	r,[@[=]]n[,x]	7-138
BMI	Branch on result code of negative, Assembler supplies the R field of 4	95	[@[=]]n[,x]	7-133
BNE	Branch on compare code of not equal, Assembler supplies R field of 6	91	[@[=]]n[,x]	7-132
BNZ	Branch on result code of not zero, Assembler supplies the R field of 6	95	[@[=]]n[,x]	7-133
BO	Branch on floating point exponent overflow, Assembler supplies R field of 2	9D	[@[=]]n[,x]	7-134
BPL	Branch on result code of positive, Assembler supplies the R field of 2	95	[@[=]]n[,x]	7-133
BRC	Branch on result code true	95	m,[@[=]]n[,x]	7-133
BRM	Branch on result code of bits mixed zeros and ones, Assembler supplies the R field of 4	95	[@[=]]n[,x]	7-133
BRNM	Branch on result code of bits not mixed zeros and ones, Assembler supplies the R field of 3	95	[@[=]]n[,x]	7-133
BRNO	Branch on result code of not all bits ones, Assembler supplies the R field of 5	95	[@[=]]n[,x]	7-133
BRNZ	Branch on result code of not all bits zeros, Assembler supplies the R field of 6	95	[@[=]]n[,x]	7-133
BRO	Branch on result code of all bits are one, Assembler supplies the R field of 2	95	[@[=]]n[,x]	7-133
BRZ	Branch on result code of all bits are zero, Assembler supplies the R field of one	95	[@[=]]n[,x]	7-133
BU	Branch on floating point exponent underflow, Assembler supplies R field of one	9D	[@[=]]n[,x]	7-134
BUO	Branch on floating point exponent underflow or overflow, Assembler supplies R field of 3	9D	[@[=]]n[,x]	7-134
BX	Branch on fixed point overflow, Assembler supplies R field of 4	9D	[@[=]]n[,x]	7-134

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
BXEC	Branch on Execute branch condition true, Assembler supplies R field of one or odd	9C	[@]n[,x]	7-135
BXO	Branch on fixed point overflow or floating point exponent overflow, Assembler supplies R field of 6	9D	[@[=]]n[,x]	7-134
BXU	Branch on fixed point overflow or floating point exponent underflow, Assembler supplies R field of 5	9D	[@[=]]n[,x]	7-134
BXUO	Branch on fixed point overflow or floating point exponent overflow or underflow, Assembler supplies R field of 7	9D	[@[=]]n[,x]	7-134
BZ	Branch on result code of zero, Assembler supplies the R field of one	95	[@[=]]n[,x]	7-133
BZM	Branch on result code of zero or negative, Assembler supplies the R field of 5	95	[@[=]]n[,x]	7-133
BZP	Branch on result code of zero or positive, Assembler supplies the R field of 3	95	[@[=]]n[,x]	7-133
C	Compare arithmetic register, fixed point singleword	C8	r,[@][=]n[,x]	7-104
C	Compare index or vector register, fixed point singleword	CE	r,[@][=]n[,x]	7-104
CAND	Compare logical AND, singleword - arithmetic register	E2	r,[@][=]n[,x]	7-110
CANDD	Compare logical AND, doubleword - arithmetic register	E3	r,[@][=]n[,x]	7-111
CANDI	Compare immediate logical AND, singleword - arithmetic register	F2	r,i[,x]	7-112
CF	Compare arithmetic register, floating point singleword	CA	r,[@][=]n[,x]	7-106
CFD	Compare arithmetic register, floating point doubleword	CB	r,[@][=]n[,x]	7-107
CH	Compare arithmetic register, fixed point halfword	C9	r,[@][=]n[,x]	7-105
CI	Compare immediate arithmetic register, fixed point singleword	D8	r,i[,x]	7-108
CI	Compare index or vector register with immediate singleword	DE	r,i[,x]	7-108
CIH	Compare arithmetic register immediate, fixed point halfword	D9	r,i[,x]	7-109

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
COR	Compare logical OR, singleword - arithmetic register	E6	r,[@][=]n[,x]	7-113
CORD	Compare logical OR, doubleword - arithmetic register	E7	r,[@][=]n[,x]	7-114
CORI	Compare immediate logical OR, singleword - arithmetic register	F6	r,i[,x]	7-115
D	Divide into arithmetic register, fixed point singleword	64	r,[@][=]n[,x]	7-69
DBNZ	Decrement arithmetic register, test, and branch on not zero	8B	r,[@[=]]n[,x]	7-125
DBNZ	Decrement index or vector register, test, and branch on not zero	8F	r,[@[=]]n[,x]	7-125
DBZ	Decrement arithmetic register, test, and branch on zero	8A	r,[@[=]]n[,x]	7-124
DBZ	Decrement index or vector register, test, and branch on zero	8E	r,[@[=]]n[,x]	7-124
DF	Divide into arithmetic register, floating point singleword	66	r,[@][=]n[,x]	7-71
DFD	Divide into arithmetic register, floating point doubleword	67	r,[@][=]n[,x]	7-72
DH	Divide into arithmetic register, fixed point halfword	65	r,[@][=]n[,x]	7-70
DI	Divide immediate into arithmetic register, fixed point singleword	74	r,i[,x]	7-73
DIH	Divide immediate into arithmetic register, fixed point halfword	75	r,i[,x]	7-74
DSE	Decrement arithmetic register, test, and skip on equal	82	r,[@][=]n[,x]	7-119
DSNE	Decrement arithmetic register, test, and skip on not equal	83	r,[@][=]n[,x]	7-120
EQC	Equivalence, singleword - arithmetic register	EC	r,[@][=]n[,x]	7-85
EQCD	Equivalence, doubleword - arithmetic register	ED	r,[@][=]n[,x]	7-86
EQCI	Equivalence immediate, singleword - arithmetic register	FC	r,i[,x]	7-87
FDFA	Convert floating point doubleword to fixed point singleword	A2	r,[@]n[,x]	7-150
FHFD	Convert fixed point halfword to floating point doubleword	AB	r,[@]n[,x]	7-154

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
FHFL	Convert fixed point halfword to floating point singleword	A9	r,[@]n[,x]	7-152
FLFH	Convert floating point singleword to fixed point halfword	A1	r,[@]n[,x]	7-149
FLFX	Convert floating point singleword to fixed point singleword	A0	r,[@]n[,x]	7-148
FXFD	Convert fixed point singleword to floating point doubleword	AA	r,[@]n[,x]	7-153
FXFL	Convert fixed point singleword to floating point singleword	A8	r,[@]n[,x]	7-151
IBNZ	Increment arithmetic register, test, and branch on not zero	89	r,[@[=]]n[,x]	7-123
IBNZ	Increment index or vector register, and branch on not zero	8D	r,[[[=]]n[,x]	7-123
IBZ	Increment arithmetic register, test, and branch on zero	88	r,[@[=]]n[,x]	7-122
IBZ	Increment index or vector register, test, and branch on zero	8C	r,[@[=]]n[,x]	7-122
INT	Interpret - arithmetic register	92	r,[@][=]n[,x]	7-162
ISE	Increment arithmetic register, test and skip on equal	80	r,[@][=]n[,x]	7-117
ISNE	Increment arithmetic register, test, and skip on not equal	81	r,[@][=]n[,x]	7-118
L	Load arithmetic register, singleword	14	r,[@][=]n[,x]	7-3
L	Load base register, singleword	18	r,[@][=]n[,x]	7-3
L	Load index register or vector parameter register, singleword	1C	r,[@][=]n[,x]	7-3
LAC	Load arithmetic exception condition	13	[@[=]]n[,x]	7-167
LAM	Load arithmetic mask	12	[@[=]]n[,x]	7-166
LD	Load arithmetic register doubleword	17	r,[@][=]n[,x]	7-7
LEA	Load effective address into base register	52	r,[@][=]n[,x]	7-160
LEA	Load effective address into index or vector register	56	r,[@][=]n[,x]	7-160
LF	Load base register file A, M=0	1B	m,[@]n[,x]	7-23
LF	Load base register file B, M=1	1B	m,[@]n[,x]	7-23
LF	Load arithmetic register file C, M=2	1B	m,[@]n[,x]	7-23
LF	Load arithmetic register file D, M=3	1B	m,[@]n[,x]	7-23

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
LF	Load index register file X, M=4	1B	m,[@]n[,x]	7-23
LF	Load vector parameter register file V, M=5	1B	m,[@]n[,x]	7-23
LFM	Load all six eight-word register files	1F	[@]n[,x]	7-24
LH	Load arithmetic register, halfword	15	r,[@][=]n[,x]	7-4
LI	Load immediate into arithmetic register singleword	54	r,i[,x]	7-8
LI	Load immediate into index register, or vector parameter register, singleword	5C	r,i[,x]	7-8
LIH	Load immediate into arithmetic register, halfword	55	r,i[,x]	7-9
LL	Load memory right halfword, indexed, into arithmetic register left halfword	19	r,[@][=]n[,x]	7-6
LLA	Load look ahead	16	i	7-159
LM	Load magnitude, fixed point singleword, arithmetic register	3C	r,[@][=]n[,x]	7-14
LMD	Load magnitude, floating point doubleword, arithmetic register	3F	r,[@][=]n[,x]	7-17
LMF	Load magnitude, floating point singleword, arithmetic register	3E	r,[@][=]n[,x]	7-16
LMH	Load magnitude, fixed point halfword, arithmetic register	3D	r,[@][=]n[,x]	7-15
LN	Load negative, fixed point singleword, arithmetic register	30	r,[@][=]n[,x]	7-10
LND	Load negative, floating point doubleword, arithmetic register	33	r,[@][=]n[,x]	7-13
LNF	Load negative, floating point singleword, arithmetic register	32	r,[@][=]n[,x]	7-12
LNH	Load negative, fixed point halfword, arithmetic register	31	r,[@][=]n[,x]	7-11
LNM	Load negative magnitude, fixed point singleword, arithmetic register	38	r,[@][=]n[,x]	7-18
LNMD	Load negative magnitude, floating point doubleword, arithmetic register	3B	r,[@][=]n[,x]	7-21
LNMF	Load negative magnitude, floating point singleword, arithmetic register	3A	r,[@][=]n[,x]	7-20
LNMH	Load negative magnitude, fixed point halfword, arithmetic register	39	r,[@][=]n[,x]	7-19
LO	Load arithmetic register with ones complement singleword	1E	r,[@][=]n[,x]	7-22

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
LR	Load memory right halfword, indexed, into arithmetic register right halfword	1D	r,[@][=]n[,x]	7-5
M	Multiply fixed point singleword - arithmetic register	6C	r,[@][=]n[,x]	7-63
M	Multiply, fixed point singleword - base register	68	r,[@][=]n[,x]	7-63
M	Multiply, fixed point singleword - index or vector parameter register	6A	r,[@][=]n[,x]	7-63
MCP	Monitor call and proceed	90	i[,x]	
MCW	Monitor call and wait	94	i[,x]	7-164
MF	Multiply, floating point singleword - arithmetic register	6E	r,[@][=]n[,x]	7-65
MFD	Multiply, floating point doubleword - arithmetic register	6F	r,[@][=]n[,x]	7-66
MH	Multiply, fixed point halfword - arithmetic register	6D	r,[@][=]n[,x]	7-64
MI	Multiply immediate, fixed point singleword - arithmetic register	7C	r,i[,x]	7-67
MI	Multiply immediate, fixed point singleword - base register	78	r,i[,x]	7-67
MI	Multiply immediate, fixed point singleword - index or vector parameter register	7A	r,i[,x]	7-67
MIH	Multiply immediate, fixed point halfword - arithmetic register	7D	r,i[,x]	7-68
MOD	Modify stack parameter doubleword	9F	r,[@]n[,x]	7-143
NFH	Normalize fixed point halfword	AD	r,[@]n[,x]	7-156
NFX	Normalize fixed point singleword	AC	r,[@]n[,x]	7-155
NOP	Take next instruction, Assembler supplies R field of zero	91	[@[=]]n[,x]	7-132
OR	OR, singleword - arithmetic register	E4	r,[@][=]n[,x]	7-79
ORD	OR, doubleword - arithmetic register	E5	r,[@][=]n[,x]	7-80
ORI	OR immediate, singleword - arithmetic register	F4	r,i[,x]	7-81
PSH	Push word into last-in-first-out stack	93	r,[@]n[,x]	7-141
PUL	Pull word from last-in-first-out stack	97	r,[@]n[,x]	7-142
RVS	Bit reversal, singleword - arithmetic register	C6	r,i[,x]	7-102
S	Subtract from arithmetic register, fixed	48	r,[@][=]n[,x]	7-53

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
SA	Arithmetic shift, fixed point singleword - arithmetic register	C0	r,i[,x]	7-93
SAD	Arithmetic shift, fixed point doubleword - arithmetic register	C3	r,i[,x]	7-95
SAH	Arithmetic shift, fixed point halfword - arithmetic register	C1	r,i[,x]	7-94
SC	Circular shift, singleword - arithmetic register	CC	r,i[,x]	7-99
SCD	Circular shift, doubleword - arithmetic register	CF	r,i[,x]	7-101
SCH	Circular shift, halfword - arithmetic register	CD	r,i[,x]	7-100
SF	Subtract from arithmetic register, floating point singleword	4A	r,[@][=]n[,x]	7-55
SFD	Subtract from arithmetic register, floating point doubleword	4B	r,[@][=]n[,x]	7-56
SH	Subtract from arithmetic register, fixed point halfword	49	r,[@][=]n[,x]	7-54
SI	Subtract immediate from arithmetic register, fixed point singleword	58	r,i[,x]	7-57
SIH	Subtract immediate from arithmetic register, fixed point halfword	59	r,i[,x]	7-58
SL	Logical shift, singleword - arithmetic register	C4	r,i[,x]	7-96
SLD	Logical shift, doubleword - arithmetic register	C7	r,i[,x]	7-98
SLH	Logical shift, halfword - arithmetic register	C5	r,i[,x]	7-97
SM	Subtract magnitude from arithmetic register, fixed point singleword	4C	r,[@][=]n[,x]	7-59
SMF	Subtract magnitude from arithmetic register, floating point singleword	4E	r,[@][=]n[,x]	7-61
SMFD	Subtract magnitude from arithmetic register, floating point doubleword	4F	r,[@][=]n[,x]	7-62
SMH	Subtract magnitude from arithmetic register, fixed point halfword	4D	r,[@][=]n[,x]	7-60
SPS	Store program status word	22	[@]n[,x]	7-168
ST	Store arithmetic register, singleword	24	r,[@]n[,x]	7-26
ST	Store base register, singleword	28	r,[@]n[,x]	7-26
ST	Store index register or vector parameter register, singleword	2C	r,[@]n[,x]	7-26

ASSMB CODE	INSTRUCTION	MCHN CODE	OPERAND FORMAT	TOPIC
STD	Store arithmetic register, doubleword	27	r,[@]n[,x]	7-30
STF	Store base register file A, M=0	2B	m,[@]n[,x]	7-39
STF	Store base register file B, M=1	2B	m,[@]n[,x]	7-39
STF	Store arithmetic register file C, M=2	2B	m,[@]n[,x]	7-39
STF	Store arithmetic register file D, M=3	2B	m,[@]n[,x]	7-39
STF	Store index register file X, M=4	2B	m,[@]n[,x]	7-39
STF	Store vector parameter register file V, M=5	2B	m,[@]n[,x]	7-39
STFM	Store all six eight word register files	2F	[@]n[,x]	7-41
STH	Store halfword, arithmetic register	25	r,[@]n[,x]	7-27
STL	Store arithmetic register left halfword into memory right halfword, indexed	29	r,[@]n[,x]	7-29
STN	Store negative, fixed point word	34	r,[@]n[,x]	7-34
STND	Store negative, floating point doubleword	37	r,[@]n[,x]	7-37
STNF	Store negative, floating point word	36	r,[@]n[,x]	7-36
STNH	Store negative, fixed point halfword	35	r,[@]n[,x]	7-35
STO	Store ones complement, word	2E	r,[@]n[,x]	7-38
STOH	Store ones complement, halfword	2A	r,[@]n[,x]	7-39
STR	Store arithmetic register right halfword into memory right halfword, indexed	2D	r,[@]n[,x]	7-28
STZ	Store zero, word	20	[@]n[,x]	7-31
STZD	Store zero, doubleword	23	[@]n[,x]	7-33
STZH	Store zero, halfword	21	[@]n[,x]	7-32
VECT	Execute vector parameter file, Assembler supplies R field of one	B0	[@]n[,x]	8-5
VECTL	Load and execute vector parameter file, Assembler supplies R field of zero	B0	[@]n[,x]	8-4
XCH	Exchange - arithmetic register with effective address	1A	r,[@]n[,x]	7-158
XEC	Execute addressed instruction in line	96	[@][=]n[,x]	7-161
XOR	Exclusive OR, singleword - arithmetic register	E8	r,[@][=]n[,x]	7-82
XORD	Exclusive OR, doubleword - arithmetic register	E9	r,[@][=]n[,x]	7-83
XORI	Exclusive OR immediate, singleword - arithmetic register	F8	r,i[,x]	7-84

Table H-3. Central Processor Scalar Instructions In Numeric Order By Machine Code

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
12	LAM	Load arithmetic mask	[@[=]]n[,x]	7-166
13	LAC	Load arithmetic exception condition	[@[=]]n[,x]	7-167
14	L	Load arithmetic register, singleword	r,[@][=]n[,x]	7-3
15	LH	Load arithmetic register, halfword	r,[@][=]n[,x]	7-4
16	LLA	Load look ahead	i	7-159
17	LD	Load arithmetic register, doubleword	r,[@][=]n[,x]	7-7
18	L	Load base register, singleword	r,[@][=]n[,x]	7-3
19	LL	Load memory right halfword, indexed, into arithmetic register left halfword	r,[@][=]n[,x]	7-6
1A	XCH	Exchange - arithmetic register with effective address	r,[@]n[,x]	7-158
1B	LF	Load base register file A, M=0	m,[@]n[,x]	7-23
1B	LF	Load base register file B, M=1	m,[@]n[,x]	7-23
1B	LF	Load arithmetic register file C, M=2	m,[@]n[,x]	7-23
1B	LF	Load arithmetic register file D, M=3	m,[@]n[,x]	7-23
1B	LF	Load index register file X, M=4	m,[@]n[,x]	7-23
1B	LF	Load vector parameter register file V, M=5	m,[@]n[,x]	7-23
1C	L	Load index register or vector parameter register, singleword	r,[@][=]n[,x]	7-3
1D	LR	Load memory right halfword, indexed, into arithmetic register right halfword	r,[@][=]n[,x]	7-5
1F	LO	Load arithmetic register with ones complement, singleword	r,[@][=]n[,x]	7-22
1F	LFM	Load all six eight-word register files	[@]n[,x]	7-24
20	STZ	Store zero, word	[@]n[,x]	7-31
21	STZH	Store zero, halfword	[@]n[,x]	7-32
22	SPS	Store program status word	[@]n[,x]	7-168
23	STZD	Store zero, doubleword	[@]n[,x]	7-33
24	ST	Store arithmetic register, singleword	r,[@]n[,x]	7-26
25	STH	Store halfword, arithmetic register	r,[@]n[,x]	7-27
27	STD	Store arithmetic register, doubleword	r,[@]n[,x]	7-30
28	ST	Store base register, singleword	r,[@]n[,x]	7-26
29	STL	Store arithmetic register left halfword into memory right halfword, indexed	r,[@]n[,x]	7-29

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
2A	STOH	Store ones complement, halfword	r,[@]n[,x]	7-39
2B	STF	Store base register file A, M=0	m,[@]n[,x]	7-40
2B	STF	Store base register file B, M=1	m,[@]n[,x]	7-40
2B	STF	Store arithmetic register file C, M=2	m,[@]n[,x]	7-40
2B	STF	Store arithmetic register file D, M=3	m,[@]n[,x]	7-40
2B	STF	Store index register file X, M=4	m,[@]n[,x]	7-40
2B	STF	Store vector parameter register file V, M=5	m,[@]n[,x]	7-40
2C	ST	Store index register or vector parameter register	r,[@]n[,x]	7-26
2D	STR	Store arithmetic register right halfword into memory right halfword, indexed	r,[@]n[,x]	7-28
2E	STO	Store ones complement, word	r,[@]n[,x]	7-38
2F	STFM	Store all six eight-word register files	[@]n[,x]	7-41
30	LN	Load negative, fixed point single word, arithmetic register	r,[@][=]n[,x]	7-10
31	LNH	Load negative, fixed point halfword, arithmetic register	r,[@][=]n[,x]	7-11
32	LNF	Load negative, floating point singleword, arithmetic register	r,[@][=]n[,x]	7-12
33	LND	Load negative, floating point doubleword, arithmetic register	r,[@][=]n[,x]	7-13
34	STN	Store negative, fixed point word	r,[@]n[,x]	7-34
35	STNH	Store negative, fixed point halfword	r,[@]n[,x]	7-35
36	STNF	Store negative, floating point word	r,[@]n[,x]	7-36
37	STND	Store negative, floating point doubleword	r,[@]n[,x]	7-37
38	LNM	Load negative magnitude, fixed point single-word, arithmetic register	r,[@][=]n[,x]	7-18
39	LNMH	Load negative magnitude, fixed point half-word, arithmetic register	r,[@][=]n[,x]	7-18
3A	LNMF	Load negative magnitude, floating point singleword, arithmetic register	r,[@][=]n[,x]	7-20
3B	LNMD	Load negative magnitude, floating point doubleword, arithmetic register	r,[@][=]n[,x]	7-21
3C	LM	Load magnitude, fixed point singleword, arithmetic register	r,[@][=]n[,x]	7-14
3D	LMH	Load magnitude, fixed point halfword, arithmetic register	r,[@][=]n[,x]	7-15

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
3E	LMF	Load magnitude, floating point singleword, arithmetic register	r,[@][=]n[,x]	7-16
3F	LMD	Load magnitude, floating point doubleword, arithmetic register	r,[@][=]n[,x]	7-17
40	A	Add to arithmetic register, fixed point singleword	r,[@][=]n[,x]	7-43
41	AH	Add to arithmetic register, fixed point halfword	r,[@][=]n[,x]	7-44
42	AF	Add to arithmetic register, floating point singleword	r,[@][=]n[,x]	7-45
43	AFD	Add to arithmetic register, floating point doubleword	r,[@][=]n[,x]	7-46
44	AM	Add magnitude to arithmetic register, fixed point singleword	r,[@][=]n[,x]	7-49
45	AMH	Add magnitude to arithmetic register, fixed point halfword	r,[@][=]n[,x]	7-50
46	AMF	Add magnitude to arithmetic register, floating point singleword	r,[@][=]n[,x]	7-51
47	AMFD	Add magnitude to arithmetic register, floating point doubleword	r,[@][=]n[,x]	7-52
48	S	Subtract from arithmetic register, fixed point singleword	r,[@][=]n[,x]	7-53
49	SH	Subtract from arithmetic register, fixed point halfword	r,[@][=]n[,x]	7-54
4A	SF	Subtract from arithmetic register, floating point singleword	r,[@][=]n[,x]	7-55
4B	SFD	Subtract from arithmetic register, floating point doubleword	r,[@][=]n[,x]	7-56
4C	SM	Subtract magnitude from arithmetic register, fixed point singleword	r,[@][=]n[,x]	7-59
4D	SMH	Subtract magnitude from arithmetic register, fixed point halfword	r,[@][=]n[,x]	7-60
4E	SMF	Subtract magnitude from arithmetic register, floating point singleword	r,[@][=]n[,x]	7-61
4F	SMFD	Subtract magnitude from arithmetic register, floating point doubleword	r,[@][=]n[,x]	7-62
50	AI	Add immediate to arithmetic register, fixed point singleword	r,i[,x]	7-47
51	AIH	Add immediate to arithmetic register, fixed point halfword	r,i[,x]	7-48

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
52	LEA	Load effective address into base register	r,[@][=]n[,x]	7-160
54	LI	Load immediate into arithmetic register singleword	r,i[,x]	7-8
55	LIH	Load immediate into arithmetic register, halfword	r,i[,x]	7-9
56	LEA	Load effective address into index or vector register	r,[@][=]n[,x]	7-160
58	SI	Subtract immediate from arithmetic register, fixed point singleword	r,i[,x]	7-57
59	SIH	Subtract immediate from arithmetic register, fixed point halfword	r,i[,x]	7-58
5C	LI	Load immediate into index register, or vector parameter register, singleword	r,i[,x]	7-8
60	A	Add to base register, fixed point singleword	r,[@][=]n[,x]	7-43
62	A	Add to index or vector parameter register, fixed point singleword	r,[@][=]n[,x]	7-43
64	D	Divide into arithmetic register, fixed point singleword	r,[@][=]n[,x]	7-69
65	DH	Divide into arithmetic register, fixed point halfword	r,[@][=]n[,x]	7-70
66	DF	Divide into arithmetic register, floating point singleword	r,[@][=]n[,x]	7-71
67	DFD	Divide into arithmetic register, floating point doubleword	r,[@][=]n[,x]	7-72
68	M	Multiply, fixed point singleword - base register	r,[@][=]n[,x]	7-63
6A	M	Multiply, fixed point singleword - index or vector parameter register	r,[@][=]n[,x]	7-63
6C	M	Multiply, fixed point singleword - arithmetic register	r,[@][=]n[,x]	7-63
6D	MH	Multiply, fixed point halfword - arithmetic register	r,[@][=]n[,x]	7-64
6E	MF	Multiply, floating point singleword - arithmetic register	r,[@][=]n[,x]	7-65
6F	MFD	Multiply, floating point doubleword - arithmetic register	r,[@][=]n[,x]	7-66
70	AI	Add immediate to base register, fixed point singleword	r,i[,x]	7-47

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
72	AI	Add immediate to index or vector parameter register, fixed point singleword	r,i[,x]	7-47
74	DI	Divide immediate into arithmetic register, fixed point singleword	r,i[,x]	7-73
75	DIH	Divide immediate into arithmetic register, fixed point halfword	r,i[,x]	7-74
78	MI	Multiply immediate, fixed point singleword - base register	r,i[,x]	7-67
7A	MI	Multiply immediate, fixed point singleword - index or vector parameter register	r,i[,x]	7-67
7C	MI	Multiply immediate, fixed point singleword - arithmetic register	r,i[,x]	7-67
7D	MIH	Multiply immediate, fixed point halfword - arithmetic register	r,i[,x]	7-68
80	ISE	Increment arithmetic register, test, and skip on equal	r,[@][=]n[,x]	7-117
81	ISNE	Increment arithmetic register, test, and skip on not equal	r,[@][=]n[,x]	7-118
82	DSE	Decrement arithmetic register, test, and skip on equal	r,[@][=]n[,x]	7-119
83	DSNE	Decrement arithmetic register, test, and skip on not equal	r,[@][=]n[,x]	7-120
84	BCLE	Branch on arithmetic register less than or equal	r,r,n	7-128
85	BCG	Branch on arithmetic register greater than	r,r,n	7-129
86	BCLE	Branch on index or vector register less than or equal	r,r,n	7-128
87	BCG	Branch on index or vector register greater than	r,r,n	7-129
88	IBZ	Increment arithmetic register, test, and branch on zero	r,[@[=]]n[,x]	7-122
89	IBNZ	Increment arithmetic register, test, and branch on not zero	r,[@[=]]n[,x]	7-123
8A	DBZ	Decrement arithmetic register, test, and branch on zero	r,[@[=]]n[,x]	7-124
8B	DBNZ	Decrement arithmetic register, test, and branch on not zero	r,[@[=]]n[,x]	7-125
8C	IBZ	Increment index or vector register, test and branch on zero	r,[@[=]]n[,x]	7-122

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
8D	IBNZ	Increment index or vector register, and branch on not zero	r,[@[=]]n[,x]	7-123
8E	DBZ	Decrement index or vector register, test, and branch on zero	r,[@[=]]n[,x]	7-124
8F	DBNZ	Decrement index or vector register, test, and branch on not zero	r,[@[=]]n[,x]	7-125
90	MCP	Monitor call and proceed	i,[,x]	7-163
91	BCC	Branch on compare code true	m,[@[=]]n[,x]	7-132
91	NOP	Take next instruction, Assembler supplies R field of zero	[@[=]]n[,x]	7-132
91	BE	Branch on compare code of equal, Assembler supplies R field of one	[@[=]]n[,x]	7-132
91	BG	Branch on compare code of greater than, Assembler supplies R field of 2	[@[=]]n[,x]	7-132
91	BGE	Branch on compare code of greater than or equal, Assembler supplies R field of 3	[@[=]]n[,x]	7-132
91	BL	Branch on compare code of less than, Assembler supplies R field of 4	[@[=]]n[,x]	7-132
91	BLE	Branch on compare code of less than or equal, Assembler supplies R field of 5	[@[=]]n[,x]	7-132
91	BNE	Branch on compare code of not equal, Assembler supplies R field of 6	[@[=]]n[,x]	7-132
91	B	Unconditional branch, Assembler supplies R field of 7	[@[=]]n[,x]	7-132
91	BCZ	Branch on compare code of all bits are zero, Assembler supplies R field of one	[@[=]]n[,x]	7-132
91	BCO	Branch on compare code of all bits are one, Assembler supplies R field of 2	[@[=]]n[,x]	7-132
91	BCNM	Branch on compare code of not mixed, Assembler supplies R field of 3	[@[=]]n[,x]	7-132
91	BCM	Branch on compare code of mixed zeros and ones, Assembler supplies R field of 4	[@[=]]n[,x]	7-132
91	BCNO	Branch on compare code of not all ones, Assembler supplies R field of 5	[@[=]]n[,x]	7-132
91	BCNZ	Branch on compare code of not all zeros, Assembler supplies the R field of 6	[@[=]]n[,x]	7-132
92	INT	Interpret - arithmetic register	r,[@][=]n[,x]	7-162
93	PSH	Push word into last-in-first-out stack	r,[@]n[,x]	7-141

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
94	MCW	Monitor call and wait	i[,x]	7-164
95	BRC	Branch on result code true	m,[@[=]]n[,x]	7-133
95	BZ	Branch on result code of zero, Assembler supplies the R field of one	[@[=]]n[,x]	7-133
95	BPL	Branch on result code of positive, Assembler supplies the R field of 2	[@[=]]n[,x]	7-133
95	BZP	Branch on result code of zero or positive, Assembler supplies the R field of 3	[@[=]]n[,x]	7-133
95	BMI	Branch on result code of negative, Assembler supplies the R field of 4	[@[=]]n[,x]	7-133
95	BZM	Branch on result code of zero or negative, Assembler supplies the R field of 5	[@[=]]n[,x]	7-133
95	BNZ	Branch on result code of not zero, Assembler supplies the R field of 6	[@[=]]n[,x]	7-133
95	BRZ	Branch on result code of all bits are zero, Assembler supplies the R field of one	[@[=]]n[,x]	7-133
95	BRO	Branch on result code of all bits are one, Assembler supplies the R field of 2	[@[=]]n[,x]	7-133
95	BRNM	Branch on result code of bits not mixed zeros and ones, Assembler supplies the R field of 3	[@[=]]n[,x]	7-133
95	BRM	Branch on result code of bits mixed zeros and ones, Assembler supplies the R field of 4	[@[=]]n[,x]	7-133
95	BRNO	Branch on result code of not all bits ones, Assembler supplies the R field of 5	[@[=]]n[,x]	7-133
95	BRNZ	Branch on result code of not all bits zeros, Assembler supplies the R field of 6	[@[=]]n[,x]	7-133
96	XEC	Execute addressed instruction in line	[@[=]]n[,x]	7-161
97	PUL	Pull word from last-in-first-out stack	r,[@]n[,x]	7-142
98	BLB	Branch and load base register with program counter	r,[@[=]]n[,x]	7-137
99	BLX	Branch and load index or vector register with program counter	r,[@[=]]n[,x]	7-138
9C	BXEC	Branch on Execute branch condition true, Assembler supplies R field of one or odd	[@]n[,x]	7-135
9D	BAE	Branch on arithmetic exception condition true	m,[@[=]]n[,x]	7-134

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
9D	BU	Branch on floating point exponent underflow, Assembler supplies R field of one	[@[=]]n[,x]	7-134
9D	BO	Branch on floating point exponent overflow, Assembler supplies R field of 2	[@[=]]n[,x]	7-134
9D	BUO	Branch on floating point exponent underflow or overflow, Assembler supplies R field of 3	[@[=]]n[,x]	7-134
9D	BX	Branch on fixed point overflow, Assembler supplies R field of 4	[@[=]]n[,x]	7-134
9D	BXU	Branch on fixed point overflow or floating point exponent underflow, Assembler supplies R field of 5	[@[=]]n[,x]	7-134
9D	BXO	Branch on fixed point overflow or floating point exponent overflow, Assembler supplies R field of 6	[@[=]]n[,x]	7-134
9D	BXUO	Branch on fixed point overflow or floating point exponent overflow or underflow, Assembler supplies R field of 7	[@[=]]n[,x]	7-134
9D	BD	Branch on divide check, Assembler supplies R field of 8	[@[=]]n[,x]	7-134
9D	BDU	Branch on divide check or floating point exponent underflow, Assembler supplies R field of 9	[@[=]]n[,x]	7-134
9D	BDO	Branch on divide check on floating point exponent overflow, Assembler supplies R field of A	[@[=]]n[,x]	7-134
9D	BDUO	Branch on divide check or floating point exponent overflow or underflow, Assembler supplies R field of B	[@[=]]n[,x]	7-134
9D	BDX	Branch on divide check or fixed point overflow, Assembler supplies R field of C	[@[=]]n[,x]	7-134
9D	BDXU	Branch on divide check of fixed point overflow or floating point exponent underflow, Assembler supplies R field of D	[@[=]]n[,x]	7-134
9D	BDXO	Branch on divide check or fixed point overflow or floating point exponent overflow, Assembler supplies R field of E	[@[=]]n[,x]	7-134
9D	BDXUO	Branch on divide check or fixed point overflow or floating point exponent overflow or underflow, Assembler supplies R field of F	[@[=]]n[,x]	7-134
9F	MOD	Modify stack parameter doubleword	r,[@]=[,x]	7-143

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
A0	FLFX	Convert floating point singleword to fixed point singleword	r,[@]n[,x]	7-148
A1	FLFH	Convert floating point singleword to fixed point halfword	r,[@]n[,x]	7-149
A2	FDFX	Convert floating point doubleword to fixed point singleword	r,[@]n[,x]	7-150
A8	FXFL	Convert fixed point singleword to floating point singleword	r,[@]n[,x]	7-151
A9	FHFL	Convert fixed point halfword to floating point singleword	r,[@]n[,x]	7-152
AA	FXFD	Convert fixed point singleword to floating point doubleword	r,[@]n[,x]	7-153
AB	FHFD	Convert fixed point halfword to floating point doubleword	r,[@]n[,x]	7-154
AC	NFX	Normalize fixed point singleword	r,[@]n[,x]	7-155
AD	NFH	Normalize fixed point halfword	r,[@]n[,x]	7-156
B0	VECTL	Load and execute vector parameter file, Assembler supplies R field of zero	[@]n[,x]	8-4
B0	VECT	Execute vector parameter file, Assembler supplies R field of one	[@]n[,x]	8-5
C0	SA	Arithmetic shift, fixed point singleword - arithmetic register	r,i[,x]	7-93
C1	SAH	Arithmetic shift, fixed point halfword - arithmetic register	r,i[,x]	7-94
C3	SAD	Arithmetic shift, fixed point doubleword - arithmetic register	r,i[,x]	7-95
C4	SL	Logical shift, singleword - arithmetic register	r,i[,x]	7-96
C5	SLH	Logical shift, halfword - arithmetic register	r,i[,x]	7-97
C6	RVS	Bit reversal, singleword - arithmetic register	r,i[,x]	7-102
C7	SLD	Logical shift, doubleword - arithmetic register	r,i[,x]	7-98
C8	C	Compare arithmetic register, fixed point singleword	r,[@][=]n[,x]	7-104
C9	CH	Compare arithmetic register, fixed point halfword	r,[@][=]n[,x]	7-105
CA	CF	Compare arithmetic register, floating point singleword	r,[@][=]n[,x]	7-106

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
CB	CED	Compare arithmetic register, floating point doubleword	r,[@][=]n[,x]	7-107
CC	SC	Circular shift, singleword - arithmetic register	r,i[,x]	7-99
CD	SCH	Circular shift, halfword - arithmetic register	r,i[,x]	7-100
CE	C	Compare index or vector register, fixed point singleword	r,[@][=]n[,x]	7-104
CF	SCD	Circular shift, doubleword - arithmetic register	r,i[,x]	7-101
D8	CI	Compare immediate arithmetic register, fixed point singleword	r,i[,x]	7-108
D9	CIH	Compare arithmetic register immediate, fixed point halfword	r,i[,x]	7-109
DE	CI	Compare index or vector register with immediate, singleword	r,i[,x]	7-108
E0	AND	AND, singleword - arithmetic register	r,[@][=]n[,x]	7-76
E1	ANDD	AND, doubleword - arithmetic register	r,[@][=]n[,x]	7-77
E2	CAND	Compare logical AND, singleword - arithmetic register	r,[@][=]n[,x]	7-110
E3	CANDD	Compare logical AND, doubleword - arithmetic register	r,[@][=]n[,x]	7-111
E4	OR	OR, singleword - arithmetic register	r,[@][=]n[,x]	7-79
E5	ORD	OR, doubleword - arithmetic register	r,[@][=]n[,x]	7-80
E6	COR	Compare logical OR, singleword - arithmetic register	r,[@][=]n[,x]	7-113
E7	CORD	Compare logical OR, doubleword - arithmetic register	r,[@][=]n[,x]	7-114
E8	XOR	Exclusive OR, singleword - arithmetic register	r,[@][=]n[,x]	7-82
E9	XORD	Exclusive OR, doubleword - arithmetic register	r,[@][=]n[,x]	7-83
EC	EQC	Equivalence, singleword - arithmetic register	r,[@][=]n[,x]	7-85
ED	EQCD	Equivalence, doubleword - arithmetic register	r,[@][=]n[,x]	7-86
F0	ANDI	AND immediate, singleword - arithmetic register	r,i[,x]	7-78

MCHN CODE	ASSMB CODE	INSTRUCTION	OPERAND FORMAT	TOPIC
F2	CANDI	Compare immediate logical AND, single-word - arithmetic register	r, i[, x]	7-112
F4	ORI	OR immediate, singleword - arithmetic register	r, i[, x]	7-81
F6	CORI	Compare immediate logical OR, singleword - arithmetic register	r, i[, x]	7-115
F8	XORI	Exclusive OR immediate, singleword - arithmetic register	r, i[, x]	7-84
FC	EQCI	Equivalence immediate, singleword - arithmetic register	r, i[, x]	7-87

Table H-4. Central Processor Vector Instructions By Logical Grouping

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VA	40	Vector add, fixed point singleword elements	8-27
VAH	41	Vector add, fixed point halfword elements	8-27
VAF	42	Vector add, floating point singleword	8-27
VAFD	43	Vector add, floating point doubleword	8-27
VAM	44	Vector add magnitude, fixed point singleword	8-28
VAMH	45	Vector add magnitude, fixed point halfword	8-28
VAMF	46	Vector add magnitude, floating point singleword	8-28
VAMFD	47	Vector add magnitude, floating point doubleword	8-28
VS	48	Vector subtract, fixed point singleword	8-29
VSH	49	Vector subtract, fixed point halfword	8-29
VSF	4A	Vector subtract, floating point singleword	8-29
VSFD	4B	Vector subtract, floating point doubleword	8-29
VSM	4C	Vector subtract magnitude, fixed point singleword	8-30
VSMH	4D	Vector subtract magnitude, fixed point halfword	8-30
VSMF	4E	Vector subtract magnitude, floating point singleword	8-30
VSMFD	4F	Vector subtract magnitude, floating point doubleword	8-30
VM	6C	Vector multiply, fixed point singleword	8-31
VMH	6D	Vector multiply, fixed point halfword	8-31
VMF	6E	Vector multiply, floating point singleword	8-31
VMFD	6F	Vector multiply, floating point doubleword	8-31
VDP	68	Vector dot product, fixed point singleword	8-32
VDPH	69	Vector dot product, fixed point halfword	8-32
VDPF	6A	Vector dot product, floating point singleword	8-32
VDPFD	6B	Vector dot product, floating point doubleword	8-32
VD	64	Vector divide, fixed point singleword	8-33
VDH	65	Vector divide, fixed point halfword	8-33
VDF	66	Vector divide, floating point singleword	8-33
VDFD	67	Vector divide, floating point doubleword	8-33
VAND	E0	Vector logical AND, singleword	8-34
VOR	E4	Vector logical OR, singleword	8-34
VXOR	E8	Vector logical Exclusive OR, singleword	8-34
VEQC	EC	Vector logical Equivalence, singleword	8-34
VANDD	E1	Vector logical AND, doubleword	8-34

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VORD	E5	Vector logical OR, doubleword	8-34
VXORD	E9	Vector Exclusive OR, doubleword	8-34
VEQCD	ED	Vector Equivalence, doubleword	
VSA	C0	Vector arithmetic shift, fixed point singleword	8-35
VSAH	C1	Vector arithmetic shift, fixed point halfword	8-35
VSAD	C3	Vector arithmetic shift, fixed point doubleword	8-35
VSL	C4	Vector logical shift, singleword	8-35
VSLH	C5	Vector logical shift, halfword	8-35
VSLD	C7	Vector logical shift, doubleword	8-35
VSC	CC	Vector circular shift, singleword	8-35
VSCH	CD	Vector circular shift, halfword	8-35
VSCD	CF	Vector circular shift, doubleword	8-35
VMG	D8	Vector merge singlewords	8-36
VMGH	D9	Vector merge halfwords	8-36
VMGD	DB	Vector merge doublewords	8-36
VO	D4	Vector order singlewords, fixed point	8-37
VOH	D5	Vector order halfwords, fixed point	8-37
VOF	D6	Vector order singlewords, floating point	8-37
VOFD	D7	Vector order doublewords, floating point	8-37
VC	D0	Vector arithmetic comparison, fixed point singleword	8-39
VCH	D1	Vector arithmetic comparison, fixed point halfword	8-39
VCF	D2	Vector arithmetic comparison, fixed point halfword	8-39
VCFD	D3	Vector arithmetic comparison, floating point doubleword	8-39
VCAND	E2	Vector logical comparison using AND, singleword	8-40
VCANDD	E3	Vector logical comparison using AND, doubleword	8-40
VCOR	E6	Vector logical comparison using OR, singleword	8-40
VCORD	E7	Vector logical comparison using OR, doubleword	8-40
VPP	DC	Vector peak, fixed point singleword	8-42
VPPH	DD	Vector peak, fixed point halfword	8-42
VPPF	DE	Vector peak, floating point singleword	8-42
VPPFD	DF	Vector peak, floating point doubleword	8-42
VL	50	Vector search for largest arithmetic element, fixed point singleword	8-44

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VLH	51	Vector search for largest arithmetic element, fixed point halfword	8-44
VLF	52	Vector search for largest arithmetic element, floating point singleword	8-44
VLFD	53	Vector search for largest arithmetic element, floating point doubleword	8-44
VLM	54	Vector search for largest magnitude, fixed point singleword	8-45
VLMH	55	Vector search for largest magnitude, fixed point halfword	8-45
VLMF	56	Vector search for largest magnitude, floating point singleword	8-45
VLMFD	57	Vector search for largest magnitude, floating point doubleword	8-45
VSS	58	Vector search for smallest arithmetic element, fixed point singleword	8-46
VSSH	59	Vector search for smallest arithmetic element, fixed point halfword	8-46
VSSF	5A	Vector search for smallest arithmetic element, floating point singleword	8-46
VSSFD	5B	Vector search for smallest arithmetic element, floating point doubleword	8-46
VSSM	5C	Vector search for smallest magnitude, fixed point singleword	8-47
VSSMH	5D	Vector search for smallest magnitude, fixed point halfword	8-47
VSSMF	5E	Vector search for smallest magnitude, floating point singleword	8-47
VSSMFD	5F	Vector search for smallest magnitude, floating point doubleword	8-47
VFLFX	A0	Vector convert floating point singleword to fixed point singleword elements	8-49
VFLFH	A1	Vector convert floating point singleword to fixed point halfword elements	8-49
VFDFX	A2	Vector convert floating point doubleword to fixed point singleword elements	8-49
VFXFL	A8	Vector convert fixed point singleword to floating point singleword elements	8-50
VFXFD	AA	Vector convert fixed point singleword to floating point doubleword elements	8-50
VFHFL	A9	Vector convert fixed point halfword to floating point singleword elements	8-50
VFHFD	AB	Vector convert fixed point halfword to floating point doubleword elements	8-50
VNFX	AC	Vector normalize fixed point singleword	8-51
VNFH	AD	Vector normalize fixed point halfword	8-51

Table H-5. Central Processor Vector Instructions In Alphabetical Order By Assembler Code

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VA	40	Vector add, fixed point singleword	8-27
VAF	42	Vector add, floating point singleword	8-27
VAFD	43	Vector add, floating point doubleword	8-27
VAH	41	Vector add, fixed point halfword	8-27
VAM	44	Vector add magnitude, fixed point singleword	8-28
VAMF	46	Vector add magnitude, floating point singleword	8-28
VAMFD	47	Vector add magnitude, floating point doubleword	8-28
VAMH	45	Vector add magnitude, fixed point singleword	8-28
VAND	E0	Vector logical AND, singleword	8-34
VANDD	E1	Vector logical AND, doubleword	8-34
VC	D0	Vector arithmetic comparison, fixed point singleword	8-39
VCAND	E2	Vector logical comparison using AND, singleword	8-40
VCANDD	E3	Vector logical comparison using AND, doubleword	8-40
VCF	D2	Vector arithmetic comparison, floating point singleword	8-39
VCFD	D3	Vector arithmetic comparison, floating point doubleword	8-39
VCH	D1	Vector arithmetic comparison, fixed point halfword	8-39
VCOR	E6	Vector logical comparison using OR, singleword	8-40
VCORD	E7	Vector logical comparison using OR, doubleword	8-40
VD	64	Vector divide fixed point, singleword	8-33
VDF	66	Vector divide floating point, singleword	8-33
VDFD	67	Vector divide floating point, doubleword	8-33
VDH	65	Vector divide fixed point, halfword	8-33
VDP	68	Vector dot product, fixed point singleword	8-32
VDPF	6A	Vector dot product, floating point singleword	8-32
VDPFD	6B	Vector dot product, floating point doubleword	8-32
VDPH	69	Vector dot product, fixed point halfword	8-32
VEQC	EC	Vector logical Equivalence, singleword	8-34
VEQCD	ED	Vector logical Equivalence, doubleword	8-34
VFDFX	A2	Vector convert floating point doubleword to fixed point singleword	8-49
VFHFD	AB	Vector convert fixed point halfword to floating point doubleword	8-50
VFHFL	A9	Vector convert fixed point half length to floating point singleword	8-50
VFLFH	A1	Vector convert floating point singleword to fixed point halfword	8-49

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VFLFX	A0	Vector convert floating point singleword to fixed point singleword	8-49
VFXFD	AA	Vector convert fixed point singleword to fixed point doubleword	8-50
VFXFL	A8	Vector convert fixed point singleword to floating point singleword	8-50
VL	50	Vector search for largest arithmetic element, fixed point singleword	8-44
VLF	52	Vector search for largest arithmetic element, floating point singleword	8-44
VLFD	53	Vector search for largest arithmetic element, floating point doubleword	8-44
VLH	51	Vector search for largest arithmetic element, fixed point halfword	8-44
VLM	54	Vector search for largest magnitude, fixed point singleword	8-45
VLMF	56	Vector search for largest magnitude, floating point singleword	8-45
VLMFD	57	Vector search for largest magnitude, floating point doubleword	8-45
VLMH	55	Vector search for largest magnitude, fixed point halfword	8-45
VM	6C	Vector multiply, fixed point singleword	8-31
VMF	6E	Vector multiply, floating point singleword	8-31
VMFD	6F	Vector multiply, floating point doubleword	8-31
VMG	D8	Vector merge singlewords	8-36
VMGD	DB	Vector merge doublewords	8-36
VMGH	D9	Vector merge halfwords	8-36
VMH	6D	Vector multiply, fixed point halfword	8-31
VNFH	AD	Vector normalize fixed point halfword	8-51
VNFX	AC	Vector normalize fixed point singleword	8-51
VO	D4	Vector order singlewords, fixed point	8-37
VOF	D6	Vector order singlewords, floating point	8-37
VOFD	D7	Vector order doublewords, floating point	8-37
VOH	D5	Vector order halfwords, fixed point	8-37
VOR	E4	Vector logical OR, singleword	8-34
VORD	E5	Vector logical OR, doubleword	8-34
VPP	DC	Vector peak, fixed point singleword	8-42
VPPF	DE	Vector peak, floating point singleword	8-42
VPPFD	DF	Vector peak, floating point doubleword	8-42

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VPPH	DD	Vector peak, fixed point halfword	8-42
VS	48	Vector subtract, fixed point singleword	8-29
VSA	C0	Vector arithmetic shift, fixed point singleword	8-35
VSAD	C3	Vector arithmetic shift, fixed point doubleword	8-35
VSAH	C1	Vector arithmetic shift, fixed point halfword	8-35
VSC	CC	Vector circular shift, singleword	8-35
VSCD	CF	Vector circular shift, doubleword	8-35
VSCH	CD	Vector circular shift, halfword	8-35
VSF	4A	Vector subtract, floating point singleword	8-29
VSFD	4B	Vector subtract, floating point doubleword	8-29
VSH	49	Vector subtract, fixed point halfword	8-29
VSL	C4	Vector logical shift, singleword	8-35
VSLD	C7	Vector logical shift, doubleword	8-35
VSLH	C5	Vector logical shift, halfword	8-35
VSM	4C	Vector subtract magnitude, fixed point singleword	8-30
VSMF	4E	Vector subtract magnitude, floating point singleword	8-30
VSMFD	4F	Vector subtract magnitude, floating point doubleword	8-30
VSMH	4D	Vector subtract magnitude, fixed point halfword	8-30
VSS	58	Vector search for smallest arithmetic element, fixed point singleword	8-46
VSSF	5A	Vector search for smallest arithmetic element, floating point singleword	8-46
VSSFD	5B	Vector search for smallest arithmetic element, floating point doubleword	8-46
VSSH	59	Vector search for smallest arithmetic element, fixed point halfword	8-46
VSSM	5C	Vector search for smallest magnitude, fixed point singleword	8-47
VSSMF	5E	Vector search for smallest magnitude, floating point singleword	8-47
VSSNFD	5F	Vector search for smallest magnitude, floating point doubleword	8-47
VSSMH	5D	Vector search for smallest magnitude, fixed point halfword	8-47
VXOR	E8	Vector logical Exclusive OR, singleword	8-34
VXORD	E9	Vector logical Exclusive OR, doubleword	8-34

Table H-6. Central Processor Vector Instructions In Numeric Order By Machine Code

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VA	40	Vector add, fixed point singleword	8-27
VAH	41	Vector add, fixed point halfword	8-27
VAF	42	Vector add, floating point singleword	8-27
VAFD	43	Vector add, floating point doubleword	8-27
VAM	44	Vector add magnitude, fixed point singleword	8-28
VAMH	45	Vector add magnitude, fixed point halfword	8-28
VAMF	46	Vector add magnitude, floating point singleword	8-28
VAMFD	47	Vector add magnitude, floating point doubleword	8-28
VS	48	Vector subtract, fixed point singleword	8-29
VSH	49	Vector subtract, fixed point halfword	8-29
VSF	4A	Vector subtract, floating point singleword	8-29
VSFD	4B	Vector subtract, floating point doubleword	8-29
VSM	4C	Vector subtract magnitude, fixed point singleword	8-30
VSMH	4D	Vector subtract magnitude, fixed point halfword	8-30
VSMF	4E	Vector subtract magnitude, floating point singleword	8-30
VSMFD	4F	Vector subtract magnitude, floating point doubleword	8-30
VL	50	Vector search for largest arithmetic element, fixed point singleword	8-44
VLH	51	Vector search for largest arithmetic element, fixed point halfword	8-44
VLF	52	Vector search for largest arithmetic element, floating point singleword	8-44
VLFD	53	Vector search for largest arithmetic element, floating point doubleword	8-44
VLM	54	Vector search for largest magnitude, fixed point singleword	8-45
VLMH	55	Vector search for largest magnitude, fixed point halfword	8-45
VLMF	56	Vector search for largest magnitude, floating point singleword	8-45
VLMFD	57	Vector search for largest magnitude, floating point doubleword	8-45
VSS	58	Vector search for smallest arithmetic element, fixed point singleword	8-46
VSSH	59	Vector search for smallest arithmetic element, fixed point halfword	8-46
VSSF	5A	Vector search for smallest arithmetic element, floating point singleword	8-46
VSSFD	5B	Vector search for smallest arithmetic element, floating point doubleword	8-46

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VSSM	5C	Vector search for smallest magnitude, fixed point singleword	8-47
VSSMH	5D	Vector search for smallest magnitude, fixed point halfword	8-47
VSSMF	5E	Vector search for smallest magnitude, floating point singleword	8-47
VSSMFD	5F	Vector search for smallest magnitude, floating point doubleword	8-47
VD	64	Vector divide, fixed point singleword	8-33
VDH	65	Vector divide, fixed point halfword	8-33
VDE	66	Vector divide, floating point singleword	8-33
VDFD	67	Vector divide, floating point doubleword	8-33
VDP	68	Vector dot product, fixed point singleword	8-32
VDPH	69	Vector dot product, fixed point halfword	8-32
VDPF	6A	Vector dot product, floating point singleword	8-32
VDPFD	6B	Vector dot product, floating point doubleword	8-32
VM	6C	Vector multiply, fixed point singleword	8-31
VMH	6D	Vector multiply, fixed point halfword	8-31
VMF	6E	Vector multiply, floating point singleword	8-31
VMFD	6F	Vector multiply, floating point doubleword	8-31
VFLFX	A0	Vector convert floating point singleword to fixed point halfword	8-49
VFLFH	A1	Vector convert floating point singleword to fixed point halfword elements	8-49
VFDFX	A2	Vector convert floating point doubleword to fixed point singleword	8-49
VFXFL	A8	Vector convert fixed point singleword to floating point singleword	8-50
VFHFL	A9	Vector convert fixed point halfword to floating point singleword	8-50
VFXFD	AA	Vector convert fixed point singleword to floating point doubleword	8-50
VFHFD	AB	Vector convert fixed point halfword to floating point doubleword	8-50
VNFX	AC	Vector normalize, fixed point singleword	8-51
VNFH	AD	Vector normalize, fixed point halfword	8-51
VSA	C0	Vector arithmetic shift, fixed point singleword	8-35
VSAH	C1	Vector arithmetic shift, fixed point halfword	8-35
VSAD	C3	Vector arithmetic shift, fixed point doubleword	8-35
VSL	C4	Vector logical shift, singleword	8-35
VSLH	C5	Vector logical shift, halfword	8-35
VSLD	C7	Vector logical shift, doubleword	8-35

ASSMB CODE	MCHN CODE	INSTRUCTION	TOPIC
VSC	CC	Vector circular shift, singleword	8-35
VSCH	CD	Vector circular shift, halfword	8-35
VSCD	CF	Vector circular shift, doubleword	8-35
VC	D0	Vector arithmetic comparison, fixed point singleword	8-39
VCH	D1	Vector arithmetic comparison, fixed point halfword	8-39
VCF	D2	Vector arithmetic comparison, floating point singleword	8-39
VCFD	D3	Vector arithmetic comparison, floating point doubleword	8-39
VO	D4	Vector order singlewords, fixed point	8-37
VOH	D5	Vector order halfwords, fixed point	8-37
VOF	D6	Vector order singlewords, floating point	8-37
VOFD	D7	Vector order doublewords, floating point	8-37
VMG	D8	Vector merge singlewords	8-36
VMGH	D9	Vector merge halfwords	8-36
VMGD	DB	Vector merge doublewords	8-36
VPP	DC	Vector peak, fixed point singleword	8-42
VPPH	DD	Vector peak, fixed point halfword	8-42
VPPF	DE	Vector peak, floating point singleword	8-42
VPPFD	DF	Vector peak, floating point doubleword	8-42
VAND	E0	Vector logical AND, singleword	8-34
VANDD	E1	Vector logical AND, doubleword	8-34
VCAND	E2	Vector logical comparison using AND, singleword	8-40
VCANDD	E3	Vector logical comparison using AND, doubleword	8-40
VOR	E4	Vector logical OR, singleword	8-34
VORD	E5	Vector logical OR, doubleword	8-34
VCOR	E6	Vector logical comparison using OR, singleword	8-40
VCORD	E7	Vector logical comparison using OR, doubleword	8-40
VXOR	E8	Vector logical Exclusive OR, singleword	8-34
VXORD	E9	Vector logical Exclusive OR, doubleword	8-34
VEQC	EC	Vector logical Equivalence, singleword	8-34
VEQCD	ED	Vector logical Equivalence, doubleword	8-34

APPENDIX I: ASSEMBLER DIRECTIVES

Table I-1. Assembler Directives

MNEMONIC	FUNCTION	GENERAL FORM
ALIGN	sets current location to a defined multipleword boundary	blank ALIGN exp1, exp2
BASE	expresses maximum number of bases allowed	blank BASE exp
COM	defines a common module	[symbol] COM [exp1][, exp2]
COMD	defines relocatable dummy section	symbol COMD [exp1][, exp2]
COPY	causes the file "sourcefile-name" to be copied in-line as source text to the assembler	blank COPY sourcefilename
DATA	generates enough fullword fields to contain the information in the operand field	[symbol] DATA exp[, ...[, exp]]
DATAH	places two values in the respective left and right halves of the generated word	[symbol] DATAH exp1, exp2
DISP	specifies maximum size of a displacement field for base and displacement addressing	blank DISP exp
DO	processes an iterative sequence of statements	[symbol] DO [exp1][, exp2] [, exp3]]]
DROP	specifies that a given base register is no longer available for addressing	blank DROP register
DUM	defines an absolute dummy section	[symbol] DUM [exp1][, exp2]
END	terminates assembly	blank END [exp]
ENTRY	establishes linkage between separately assembled programs	blank ENTRY symbol [, ... [symbol]]
EQU	defines permanent value and section number of a label	symbol EQU exp

Table I-1. Assembler Directives (Continued)

MNEMONIC	FUNCTION	GENERAL FORM
EXTRN	declares symbols not currently defined	blank EXTRN symbol [, . . . [, symbol]]
FLAG	prints specified character at left hand of listing	blank FLAG character, exp
FORM	specifies arbitrary data format	symbol FORM exp1, exp2 [, exp3, . . . , expn]
GOTO	transfers control to indicated statement	blank GOTO symbol
IND	generates an indirect address constant	[symbol] IND exp1 [, exp2 [, exp3]]
LIST	causes listing to resume	blank LIST blank
LITORG	sets location of origin for literals	[symbol] LITORG [exp]
NAME	specifies additional names of a procedure; follows a PROC directive	symbol NAME [exp][,exp][, . . .]
NOLIST	suppresses listing	blank NOLIST blank
ORG	sets location of origin for selection being assembled	[symbol] ORG exp
PEND	terminates a procedure definition	blank PEND symbol
PEX	allows immediate exit from a procedure	blank PEX [exp]
PROC	names a procedure	symbol PROC [exp][, exp][, . . .]
RES	reserves space	[symbol] RES exp
SEC	defines control section and section number of label	[symbol] SEC [exp1][, exp2]
SET	sets temporary value and section number of label	symbol SET exp
SIZE	specifies the number of bits required to represent the object computer address	blank SIZE exp
SKIP	skip lines or eject page of listing	blank SKIP [exp[, character string]]

Table I-1. Assembler Directives (Continued)

MNEMONIC	FUNCTION	GENERAL FORM
SNAME	assign control name to next statement	symbol SNAME blank
UNIT	specifies the number of bits in the object computer word	blank UNIT exp
USING	sets a value in a specified base register	blank USING exp, register

APPENDIX J: ASSEMBLER RESTRICTIONS

J-1. LANGUAGE RESTRICTIONS

1. The maximum number of characters defined by the SET or EQU directive is 28.
2. The maximum number of characters in a string is that number which can be contained in the operand field of one statement and two continuation cards.
3. The maximum nesting of DO directives is 32.
4. The maximum level of Assembler processing (that is the depth of procedures within procedures) is 32.
5. The maximum number of EXTRN directives allowed is 255.
6. The maximum number of ENTRY directives is 255.
7. The maximum number of items within a set of parentheses is 15. (Refer to Topic 2-21).
8. The maximum level of parentheses is five. (Refer to Topic 2-21).
9. The maximum number of Control sections allowed is 63.

TEXAS INSTRUMENTS
INCORPORATED
EQUIPMENT GROUP
AUSTIN, TEXAS

PUBLICATION UPDATE

PUBLICATION

PROGRAM ASC PUBLICATION NO. S1007P

TITLE ~~Programmer's Guide to Procedure~~ Programming

DATE June 1971 JOB NO. 1158

TYPE OF CHANGE

☐ IMMEDIATE
(MAY CAUSE PERSONAL INJURY OR EQUIPMENT DAMAGE/FAILURE)

☐ ROUTINE
(BATCH PROCESSED)

SUBMITTED BY

NAME _____ PHONE _____

ADDRESS _____

MAIL STATION _____ DATE _____

LIST PAGE AND PARAGRAPH OR FIGURE NUMBERS AND DESCRIBE RECOMMENDED CHANGES.

FORWARD CHANGES BY FOLDING THIS SHEET AND STAPLING. RETURN ADDRESS IS ON BACK OF SHEET.