CDC {R} CYBER 200

MODEL INDEPENDENT INSTRUCTION SPECIFICATION

	CPU Desgin I Manager	Advanced Design Manager	Product Engineering Manager	Engineering Director
	R. C. KORT	L.K.STEINER	D.A.HANDY	L.F.GLAESER
	R.C. Kort	L K Steiner	Dale Handy	L.F. Glaeser 1/3/80

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1.0 SCOPE

This is a model independent CPU specification for the CYBER 200 line. Section 2.0 lists specification numbers for each model where information that is model dependent can be obtained from the functional specification.

This is NOT a reference manual for user's groups. This document is written expressly for logic designers and diagnostic programmers.

1.1 Definition of Radix and Power Notation

FORTRAN notation is used to indicate numbers raised to a power. For example, 2 raised to the 47th power would be written 2**47.

The following method is used to indicate the radix of numbers. The number will be followed by a radix indicator enclosed in brackets with "B" indicating binary or base 2, "D" indicating decimal or base 10, and "H" indicating hexadecimal or base 16.

For example:

100[D] = 64[H] = 1100100[B]

2.0 APPLICABLE DOCUMENTS

Model 205 - 10358025 Functional Computer Specification

- 10358026 Timing Specification

3.0 PERFORMANCE REQUIREMENTS

3.1 General Description

3.1.1 Instruction Formats and Types

3.1.1.1 Instruction Formats - all fields are 8 bits unless otherwise specified.

3.1.1.1.1 Format 1

F	G	X	A	Y	B	Z	C
Function	Sub-Function	offset for A	len. & base address	offset for B	len. & base address	C.V. base address	len. & base address

3.1.1.1.2 Format 2

F	G	X	A	Y	B	Z	C
Function	Sub-Function	O.V. len. & base	base address	O.V. len. & base	base address	O.V. len. & base	result len. & base

3.1.1.1.3 Format 3

F	G	X	A	Y	B	Z	C
Function	Sub-Function	index for A	len. & base	index for B	len. & base	index for C	len. & base

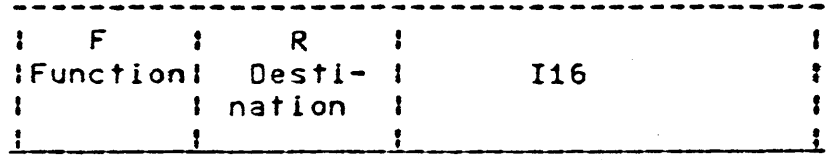
3.1.1.1.4 Format 4

F	R	S	T
function	source 1	source 2	destination

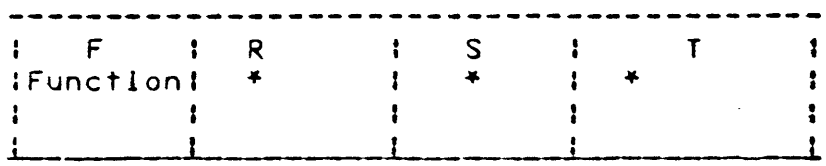
3.1.1.1.5 Format 5

F	R	
Function	destination	I48

3.1.1.1.6 Format 6

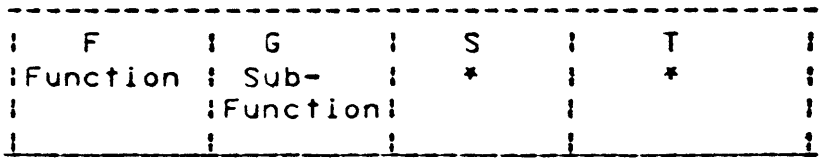


3.1.1.1.7 Format 7

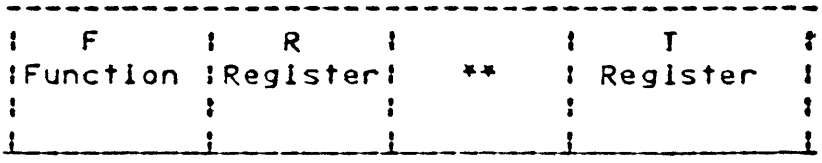


3.1.1.1.8 Format 8

3.1.1.1.9 Format 9



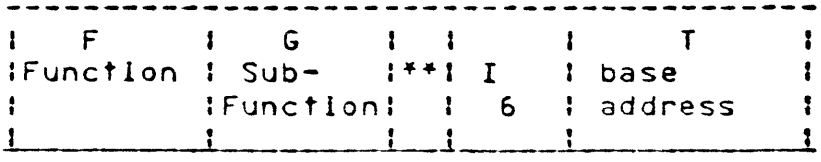
3.1.1.1.10 Format A



* Described where used

3.1.1.1.11 Format B

** Unused areas must be cleared to zeros



3.1.1.1.12 Format C

F	G	X	A	Y	B	Z	C
Function	Sub-Function	Reg-ister	Reg-ister	Index	Base Address	Reg-ister	Reg-ister

3.1.1.2 Instruction Types

3.1.1.2.1 Register Instructions (RG)

In the register instructions, all operand sources and all result destinations are registers. R, S, and T each designate the contents of one of 256 registers.

A register may be used to hold one or both source operands as well as the result. Special case: if register 00 is designated as a source or result register, see A.2 Section 3.1.7.

Unless stated differently in the instruction description in all register-to-register operations, the contents of the source registers are unchanged and the destination register is cleared before the result is transferred into it.

3.1.1.2.2 Index Instructions (IN)

The index instructions are used primarily in performing numerical calculations on field lengths and addresses.

The term, replace, means replace only the specified bits. The phrase, replace the right-most 48 bits ..., implies that the left-most 16 bits are not altered.

3.1.1.2.3 Branch Instructions (BR)

Branch conditions may be determined by examining single bits, a 24-bit or 48-bit integers, 32-bit or 64-bit integers, 32-bit floating point operands or 64-bit floating point operands. A special branch is provided to enter and leave the Monitor program. All

Item counts in branch instructions are in half-words.

3.1.1.2.4 Vector Instructions (VT)

The vector instructions perform operations on ordered scalars.

Referring to format 1 under section 3.1.1.1.1, the vector instruction designators are defined as follows:

F - eight-bit instruction code

G - eight-bit sub-operation code

X,Y - eight-bit designators, each specifying one of 256 registers holding address offsets for the source operand fields.

A,B - eight-bit designators, each specifying one of 256 registers holding base addresses and field lengths of the source operand fields.

Z - eight-bit designator specifying one of 256 registers holding the base address of the control vector.

C - eight-bit designator specifying one of 256 registers holding the base address and the field length of the destination field. If C+1 is used by the instruction, C must be even, since C+1 is formed by using the left-most seven bits of the C designator and forcing the right-most bit to a one. If C+1 is used and C is odd, the reference to C and C+1 is undefined.

C+1 - eight-bit designator specifying a register which holds the offset for both the control vector and the destination field. C+1 always references an odd register. See the preceding paragraph. Note that the usage of C+1 is dependent upon bit 2 of the G designator.

(Continued)

3.1.1.2.4 (Cont'd)

The bits of the G field (numbered from left to right as are all fields) are interpreted as follows:

Bit No.	State	Description
0	0	64-bit operands (words)
	1	32-bit operands (half-words)
1	0	*control vector operates on binary ones
	1	*control vector operates on binary zeros
2	0	do not offset destination field and control vector
	1	offset destination field and control vector
3	0	normal source stream A
	1	**broadcast repeated (A)
4	0	normal source stream B
	1	**broadcast repeated (b)
5 6 7		These bits are used for sign control for some of the Vector and Vector Macro instructions. See Section 3.1.4.9 for a description of the use of these bits for sign control. The G bit charts in the Table of Contents provide a quick reference to which instructions provide this feature.

*If the eight-bit designator Z is zero, no control vector is used, so bit 1 of G is undefined.

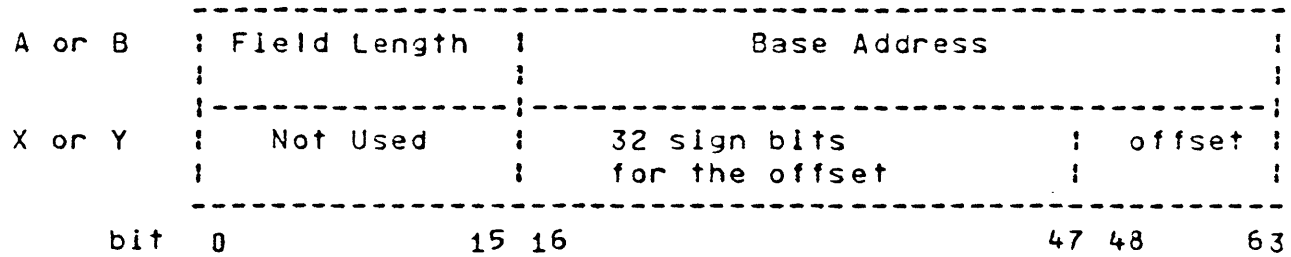
**If bit 3 and/or 4 of G is a 1, then either the A and/or B source field is a constant used as each element of the respective vector stream and the associated offsets are ignored. These constants are found in the registers specified by A and B, respectively. If bit 3 and/or 4 is a one and bit 0 of G is a one, register A and/or B is a 32-bit register. The result of broadcasting both repeated constants A and B is undefined for instructions which do not terminate due to filling the result field, i.e., the Select instructions, C0, C1, C2, and C3.

(Continued)

3.1.1.2.4 (cont'd)

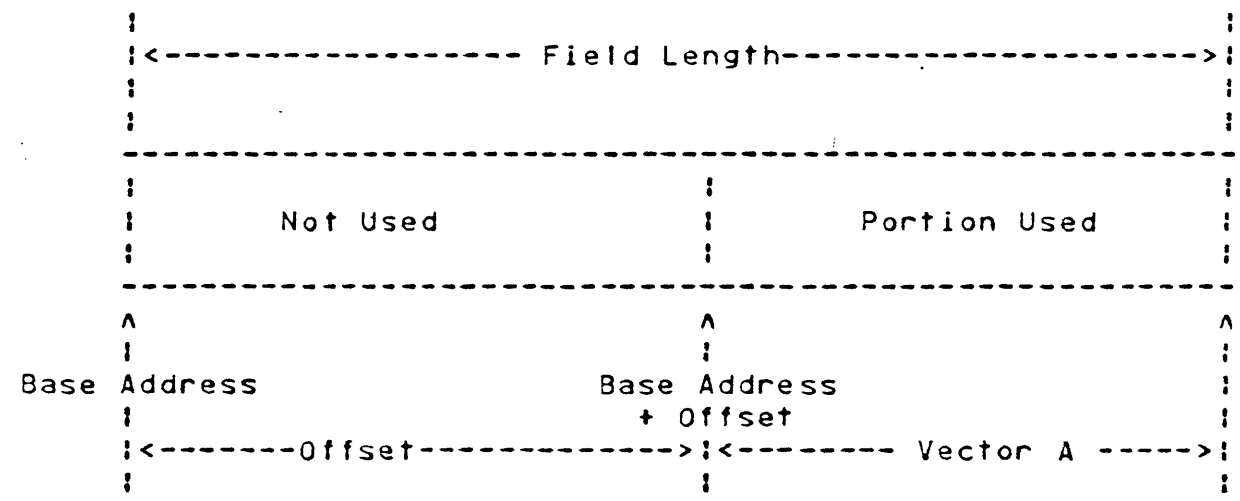
The base address (which defines the location of the first operand of vector) and the vector field lengths are obtained in the manner shown below.

The registers addressed by A and X contain the following:



If the offset does not contain 32 leading sign bits, the instruction is undefined.

The portion of the vector which would be included in vector stream A is as follows:



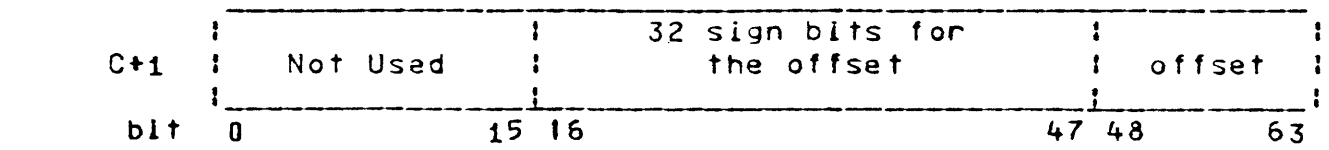
The operation of subtracting the offset from the field length must result in a vector length which is positive and less than (2**16) in magnitude. If the resulting vector length is not positive and less than (2**16) in magnitude, it will be treated as a zero vector length.

(Continued)

3.1.1.2.4 (cont'd)

The register addressed by C contains the following:

If vector C is specified as having an offset (bit 2 of the G designator is a one), register C+1 contains the offset.



If C+1 is used in the execution of an instruction, C should be specified as an even register. If C is odd, the reference to C and C+1 is undefined.

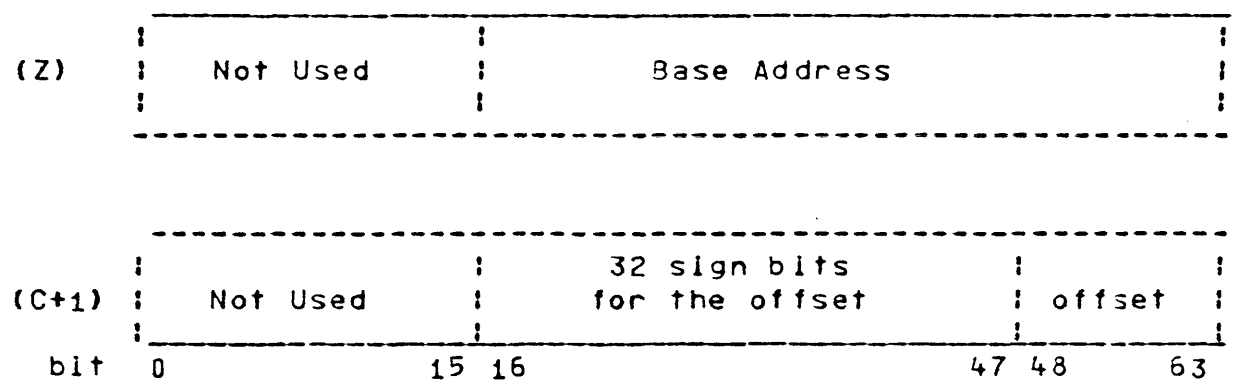
Control Vector

When control vectors are specified (Z designator ≠ 0), a single unique bit from the control vector is associated with the storing of each result element in the output field and the setting of the data flag for that result. When a bit within a control vector prohibits the storing of a result element, the previous contents of the associated result vector element are not altered nor is the data flag register modified. The nth bit read from the control vector prohibits or allows the storing of the nth result into the result vector. Bit one of the G designator selects whether a zero or a one control vector bit allows the storing of a result. If bit one of the G designator is a zero/one, store the nth result if the nth bit of the control vector was a one/zero, respectively.

(Continued)

3.1.1.2.4 (Cont'd)

Registers Z and C+1 contain the following information relating to the control vector:



Control vector Z uses the same field length as result vector C.

The starting address of the control vector is obtained by adding the offset and the base address.

Since offsets are item counts, the same offset is used for both the result vector and for the control vector. The offset indicates a bit offset when used with the control vector.

3.1.1.2.5 Vector Macro Instructions (VM)

Vector macro instructions perform in much the same manner as vector instructions.

Some vector macro instructions do not form result vectors but store their result in one or two registers which are specified by the instructions. For these instructions, the control vector has neither length nor offset and controls the use of element(s) of the source vector(s); also, bit 2 of the G field is undefined and must be set to zero. Note that C and C+1 designate 32-bit registers when bit 0 of the G designator specifies 32-bit operands.

For the other vector macro instructions (those having result vectors), the control vector has the same connotation as in vector instructions. The B7 and BA instructions do not use control vectors.

3.1.1.2.6 Sparse Vector Instructions (SV)

Due to arithmetic reduction, many elements of a vector may be reduced to zero; therefore, except for their positional significance, they need not be carried along as floating point numbers. In order to conserve both storage space and calculating time, a group of instructions make possible the expansion and compression of vectors of this type; i.e., sparse vectors.

A sparse vector consists of a vector pair, one of which is a bit string, identified as the order vector, and the other is a floating point array identified as the data vector.

A sparse vector is typically formed by first using the Compare instructions to generae an order vector. A normal vector with "near zero" elements in it is then reduced to a sparse vector with the Compress instruction. The Compress uses the generated order vector as a means to throw out all "near zero" elements. See the instruction descriptions for BC, C4, C5, C6 and C7. BC is the Compress and C4-C7 are Compare instructions.

A sparse data vector, being simply an ordered set of floating point scalars, is indistinguishable in format from any other vector. However, a sparse data vector has an associated sparse order vector which determines the positional significance of the elements of the sparse data vector. For example, a sparse data vector A and its associated sparse order vector X may be represented as follows:

near zero	a 1	a 2	near zero	near zero	a 3	a 4	near zero	Original Vector

Data Vector A	a 1	a 2	a 3	a 4	
Sparse Vector	^	^	^	^	
Order Vector X	0	1	1	0	0	1	1	0

(continued)

3.1.1.2.6 (Cont'd)

The sub-operation code and eight-bit designators have the following meanings for sparse vector instructions (see Section 3.1.1.1.2 for sparse vector format):

F - eight-bit instruction code
G - eight-bit sub-operation code - bit 0 of the G field is used in this set of instructions as follows:

State	Interpretation
0	64-bit operands
1	32-bit operands

Bits 5, 6 and 7 of the G field are used for sign control. See Section 3.1.4.9 for descriptions of the use of these bits with the above instructions.

G bits 1 and 2 are used to select the logical operation to be performed on the order vectors X and Y to form order vector Z. Bits 3 and/or 4 of the G field, when set to one, are used to broadcast A and/or B, respectively.

A,B - eight-bit designators, each specifying one of the 256 registers holding the base address of a source sparse data vector.

X,Y - eight-bit designators, each specifying one of the 256 registers containing the base address and the field length of the source sparse order vectors associated with source sparse data vectors A and B, respectively.

C - eight-bit designator specifying one of the 256 registers containing the base address of the result sparse data vector.

Z - eight-bit designator specifying one of the 256 registers containing the base address and the field length of the result sparse order vector associated with result sparse data vector C.

(continued)

3.1.1.2.6 (Cont'd)

	Not Used*	Base Address
(A),(B) or (C)	0 15 16	63

	Field Length	Base Address
(X),(Y) or (Z)	0 15 16	63

* At the completion of these instructions, the length of the resulting sparse data vector is placed in the left-most 16 bits of register C.

Neither offsetting nor indexing is performed by the sparse vector instructions. The field lengths associated with source sparse data vectors A and B are not used. These lengths are determined by the number of ones in their sparse order vectors. The field lengths of the source sparse order vectors X and Y and the result sparse order vector Z are item counts in bits.

3.1.1.2.7 String Instructions (ST)

The string instructions perform manipulations on strings of eight-bit bytes.

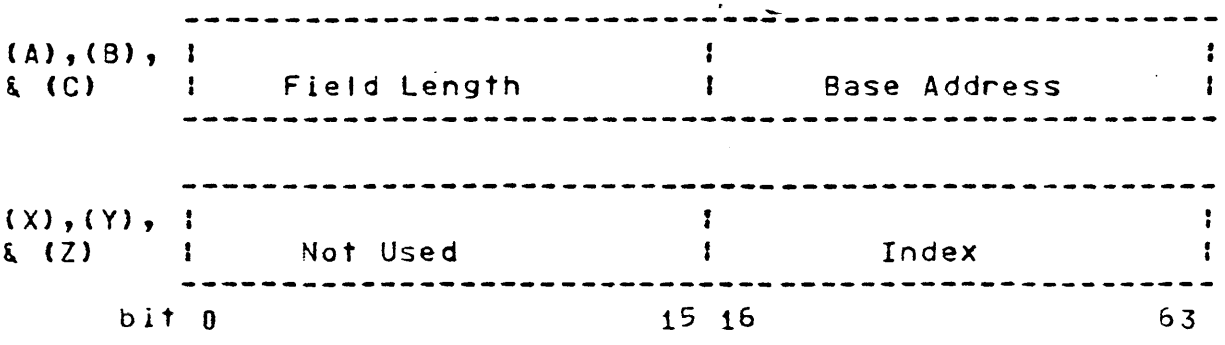
Instruction Format

The string instructions use Format 3 (see Section 3.1.1.1.3).

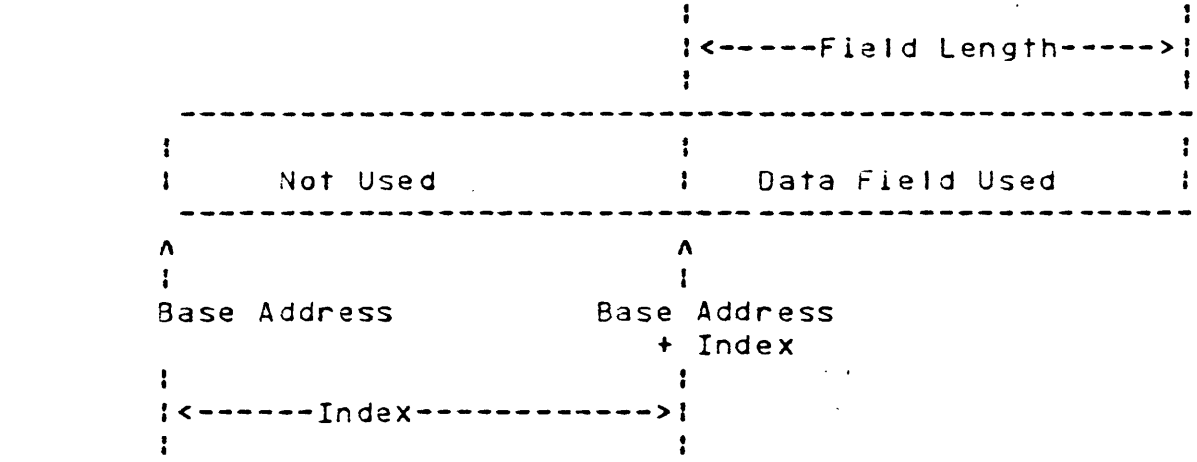
F - eight-bit instruction code
G - eight-bits unused
X,Y,Z
A,B,C - eight-bit register designators; the registers contain addressing information for the fields to be used.

(continued)

3.1.1.2.7 (Cont'd)



If any of the eight-bit designators, X, Y, or Z, are set to zero, indexing is not used for that stream and the address of the initial byte is obtained from the base address.



Note that the length of the data field used is the same as the field length found in the register containing the base address. Indexing does not affect the field length used whereas offsetting does (see offsetting in vectors 3.1.1.2.4) The string instructions do not have offsetting and the vector instructions do not have indexing.

3.1.1.2.8 Logical String (LS)

The LS (logical string) instructions have indices and fields identical to those of the ST (string) instructions except that the item counts and indices are in bits instead of bytes. The LS operations are performed as bit operations on bit boundaries while the ST operations are performed as byte operations on byte boundaries.

3.1.1.2.9 Monitor Instructions (MN)

Monitor instructions perform as described only when in Monitor Mode. When not in Monitor Mode, the Monitor instructions perform as an illegal instruction would (see Section 3.1.4.2.2).

3.1.1.2.10 Non-Typical Instruction (NT)

The format and operation of these instructions are completely described under the individual instruction write-ups.

3.1.2 Operand Size Definition and Addressing

The following operand definitions are implied throughout the specification.

Word - A 64-bit quantity, the address of the left-most bit always being a multiple of 64 base 10.

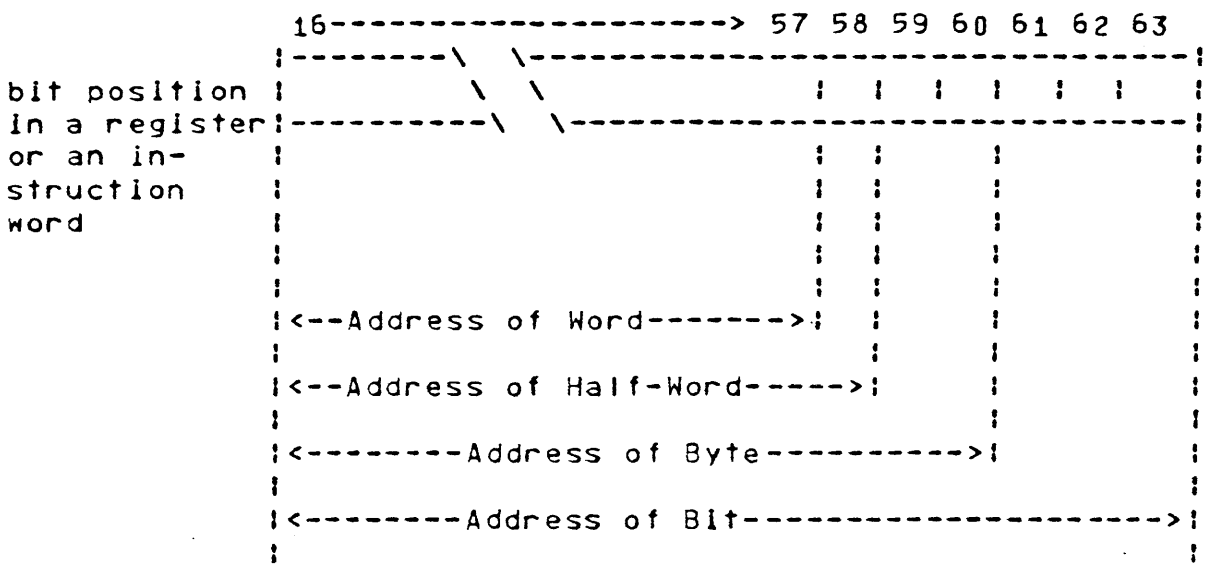
Half-word - A 32-bit quantity, the address of the left-most bit always being a multiple of 32 base 10.

Byte - An 8-bit quantity, the address of the left-most bit always being a multiple of 8 base 10.

Groups of bits in an address should be thought of as addressing various units of storage as illustrated in the chart below.

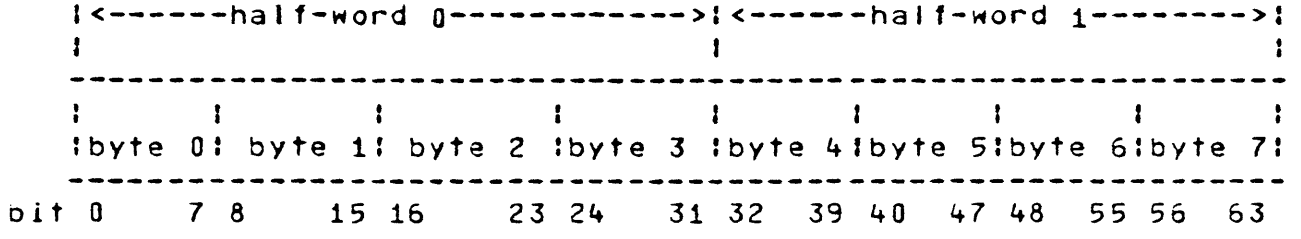
(continued)

3.1.2 (Cont'd)



Within a word, bits, bytes, and half-words are always numbered from left to right. The lowest addressed bit, byte, or half-word is always the left-most bit, byte, or half word in the word.

All addresses are 48-bit quantities and contain enough information to reference a specific bit. Depending on the usage of an address, a certain number of the right-most bits in the address are ignored. For example, if a byte is being read, the right-most three bits of the address being used to reference it are ignored. Depending on the instruction, operands are counted on a bit, byte, half-word or word basis.



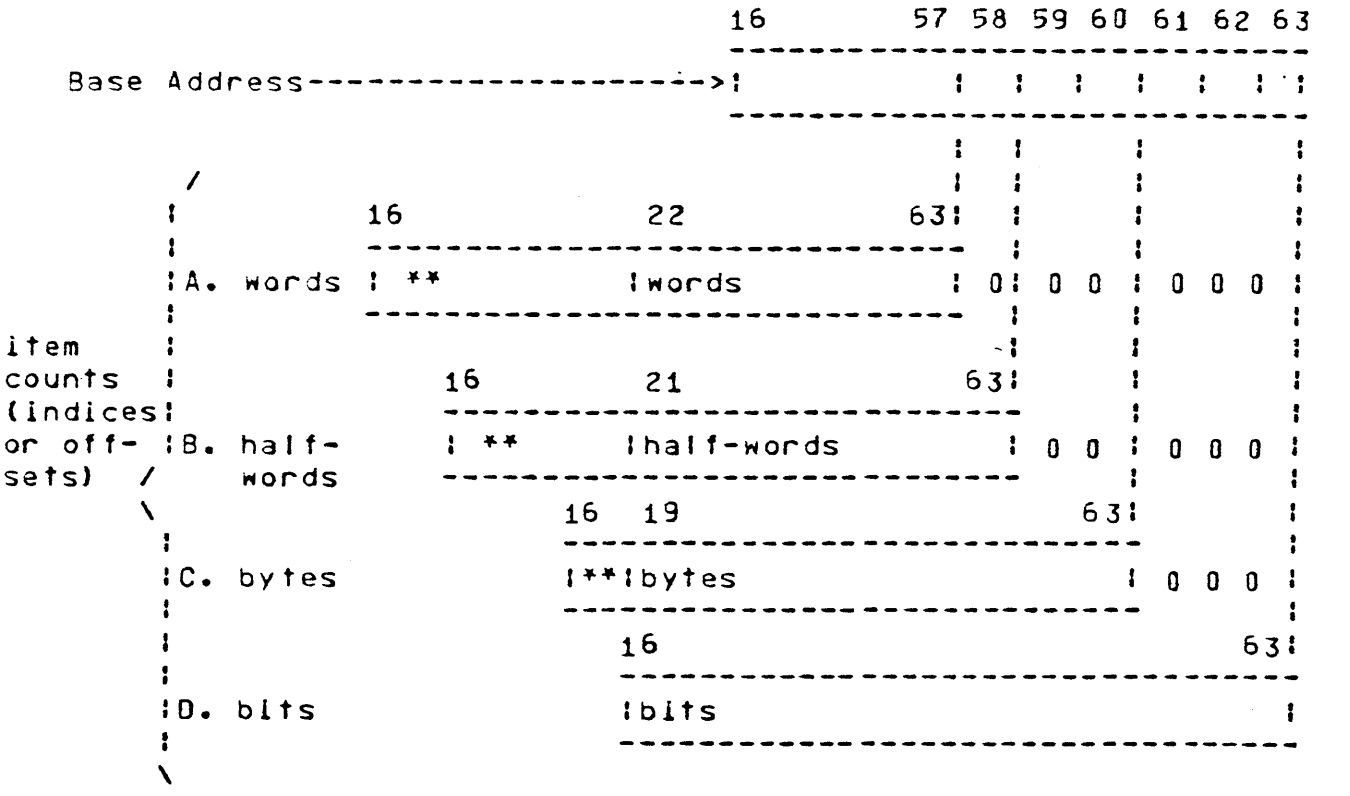
The above figure illustrates the relative location of each bit, byte and half-word within a 64-bit word.

(Continued)

3.1.2 (Cont'd)

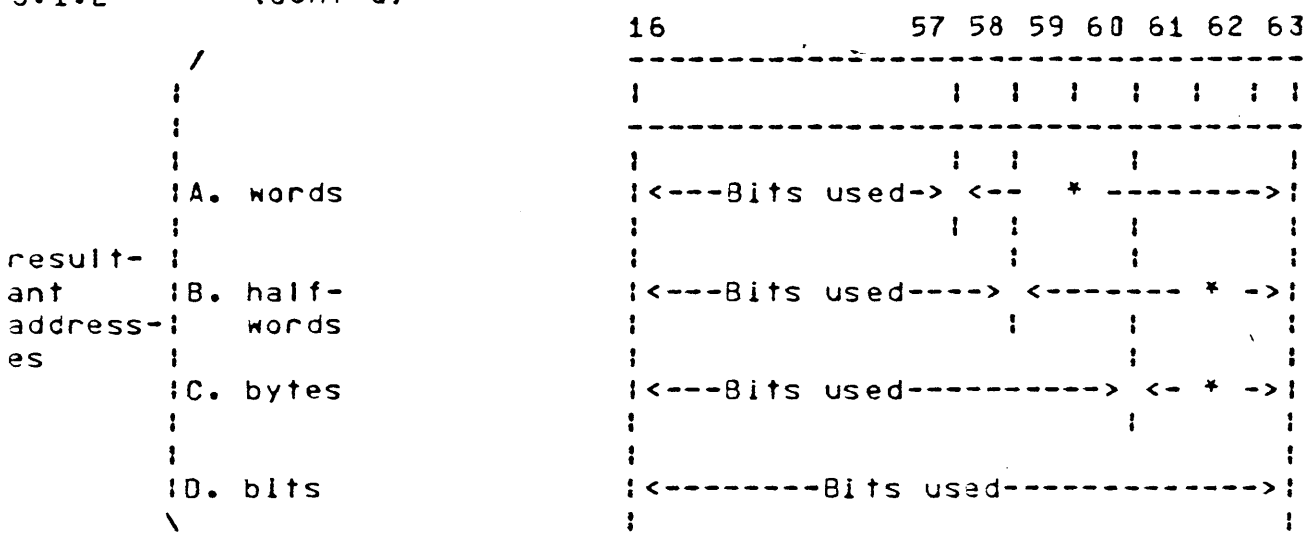
If it is necessary to add addresses and item counts (indices or offsets), the item count is shifted left end off until it is properly aligned with the address. Binary zeros are attached to the right end of the quantity being shifted.

The result of the addition always addresses a quantity having the same unit as the item count. for instance, if a byte count is added to any address, the result references a byte. This means that the right-most three bits of the address will be ignored. The following chart summarizes the process of adding an item count to an address and shows which bits are ignored in the resulting address.



(continued)

3.1.2 (Cont'd)



* These bits in the resultant address are ignored.

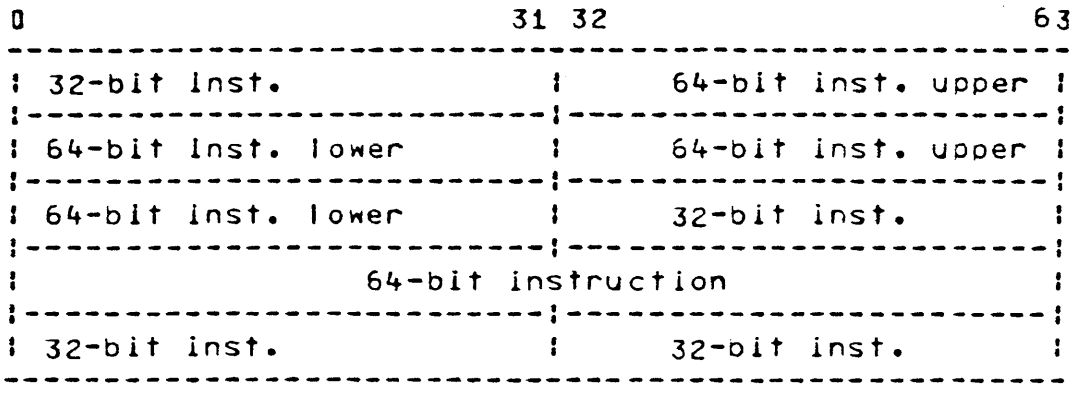
** These bits in the index or offset are shifted off and do not enter the address calculation.

The registers associated with any job or the monitor reside in the first 256 64-bit words of its associated virtual space or absolute memory, respectively. References to these portions of memory will cause the instruction to be treated as illegal in either monitor or job mode. The only exceptions to this rule are the 87 and 8A instructions with G-bit 7 set. In this case the output vector C (for the 87 instruction) or the input vector B (for the 8A instruction) must be contained in bit addresses 0 through 3FFF.

Instructions are addressed on full word and half-word boundaries. The instruction address counter will, therefore, be incremented by a half-word after executing a 32-bit instruction and by a full word after executing a 64-bit instruction. This allows instructions to be packed contiguously in storage. The following chart illustrates the various ways instructions may be packed within 64-bit words.

(continued)

3.1.2 (Cont'd)



Note that a branch is possible to any of the instructions. The lower 5 bits in any branch address will always be interpreted as zeros.

3.1.3 Termination Rules

For instructions which terminate upon exhausting the length of a data field, data string or vector: if that item is exhausted prior to the first operand fetch, the instruction becomes a no op; no data is fetched and no data flags are altered.

1. Exhausting a vector which has an offset.

   A vector is deemed exhausted prior to the first operand fetch if the result of subtracting the offset from the field length is zero or negative.

   For cases of zero field length, the resulting vector length used is the right-most 16 bits of the two's complement of the offset. If this 16-bit quantity is zero or negative, the vector is deemed exhausted prior to the first operand fetch.

   A vector is exhausted when the result of subtracting both the offset and the number of operands encountered thus far ... from the field length zero.

(continued)

3.1.3 (Cont'd)

2. Exhausting a vector which has no offset and exhausting other data fields or data strings.

   The string, field or vector is deemed exhausted prior to the first operand fetch if its length is zero or if the result of subtracting the offset from the field length is zero or negative. These strings, fields and vectors are exhausted when the result of subtracting the number of elements encountered thus far from the field length is zero.

3.1.3.1 Vector Instruction Termination

Vector instructions terminate when the result vector, vector C, is exhausted. Source vectors which are exhausted before the result vector is exhausted are extended, as required, with machine zeros in additive operations or normalized ones in multiplication or division operations.

3.1.3.2 Vector Macro Instruction Termination

Vector macro instructions with result fields (as opposed to result registers) extend short source fields with machine zeros or normalized ones and terminate in a fashion identical to the vector instructions. The B7 and BA instructions do not use extension and terminate upon exhaustion of the index field. The other vector macro instructions do not extend short source fields, but instead, terminate when either source field is exhausted. For vector macro instructions of this type, i.e., the Select instructions C0, C1, C2 C3, and DC broadcasting both source fields cause an undefined condition to exist.

3.1.3.3 Sparse Vector Instruction Termination

Sparse vector instructions terminate when order vector Z (the result order vector) is exhausted. If the Z designator is zero or if the Z length is zero, no data flags are set and the instruction is a no op. Zero length or short source order vectors are extended, as required, with zero bits. If order vector Z has a non-zero length and the C designator is zero, the results of the instruction are undefined and an illegal operand will occur if a store into C vector is required.

3.1.3.4 String Instruction Termination

String instructions terminate when the result string C is exhausted, source strings that are shorted than the result string C are extended with zeros unless otherwise specified.

3.1.3.5 Tables of Termination/Instruction Type/Field Type

M-Zero = Machine Zero,
N-One = Normalized One
No-Op = No Operation
NA = Not Applicable
I = Input
O = Output

String Instruction Terminating Conditions

Instruction Code	A Field (I)			C Field (O)	
	Result if A field is exhausted	Type of extension if any	A field length initially zero	Result if C field is exhausted	C Field length initially zero
F8	Extend	B designator Byte	Extend	Terminate	No-Op

Logical String Instruction Terminating Conditions

Instruction Code	A Field (I)			B Field (I)			C Field (O)	
	Result if A field is exhausted	Type of extension if any	A field length initially zero	Result if B field is exhausted	Type of extension if any	B field length initially zero	Result if C field is exhausted	C field length initially zero
F0,F1,F2 F3,F4,F5 F6,F7	Extend	Zero Bits	Extend	Extend	Zero Bits	Extend	Terminate	No-Op

Sparse Vector Instruction Terminating Conditions

Instruction Code	A Field (I)			B Field (I)			C Field (O)	
	Result if A/X field is exhausted	Type of extension if any	A/X field length initially zero	Result if B/Y field is exhausted	Type of extension if any	B/Y field length initially zero	Result if C/Z field is exhausted	C/Z field length initially zero
A0,A1,A2	NA	NA	NA	NA	NA	NA	NA	NA
A4,A5,A6 A8,A9,AB AC,AF	X Field (I)			Y Field (I)			Z Field (O)	
	Extend	Zero Bits	Extend	Extend	Zero Bits	Extend	Terminate	No-Op

3.1.3.5 (cont'd)

VECTOR INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field {I}			B Field {I}			C Field {0}		
	Result if A field is exhausted	Type of extension if any	A field length initially zero	Result if B field is exhausted	Type of extension if any	B field length initially zero	Result if C field is exhausted	C field length initially zero	Control Vector
80, 81, 82 83, 84, 85 86, 87 8A	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
88, 89, 8B 8C & 8F	Extend	N-One	Extend	Extend	N-One	Extend	Terminate	No-Op	Yes
90, 91, 92 & 93	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes
94 & 95	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
96, 97, 98 99 & 9A	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes
9B & 9D	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes
9C	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes

VECTOR MACRO INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field {I}			B Field {I}			C Field {0}		
	Result if A field is exhausted	Type of extension if any	A field length initially zero	Result if B field is exhausted	Type of extension if any	B field length initially zero	Result if C field is exhausted	C field length initially zero	Control Vector
B7	Terminate	NA	No-Op	NA	NA	NA	NA	NA	No
B8	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes (0)
BA	Terminate	NA	No-Op	NA	NA	NA	NA	NA	No
C0, C1, C2 & C3	Terminate *	NA	No-Op *	Terminate	NA	No-Op	NA	NA	Yes (I)
D0 & D4	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate	No-Op	Yes (0)
D1 & D5	Extend	M-Zero	Extend	NA	NA	NA	Terminate	No-Op	Yes (0)
DA & DB	Terminate	NA	No-Op	NA	NA	NA	NA	NA	Yes (I)
DC	Terminate	NA	No-Op	Terminate	NA	No-Op	NA	NA	Yes (I)
DE	Extend	N-One	Extend	Exit Loop	NA	No-Op	Terminate	No-Op	Yes (0)
DF	NA	NA	NA	NA	NA	NA	Terminate	No-Op	Yes (0)

*These instructions may terminate for reasons other than the exhausting of field length.

3.1.3.5 (Cont'd)

TERMINATING CONDITIONS FOR NONTYPICAL (32-BIT FORMAT) INSTRUCTIONS HAVING MULTIPLE OPERANDS

Instruction Code	R Field {I}		S Field {I}		T Field {O}	
	Result if R Field is exhausted	R Field length initially zero	Result if S Field is exhausted	S Field length initially zero	Result if T Field is exhausted	T Field length initially zero
14	Exit Loop	No-Op	Exit Loop	Zero R bits Skipped	Terminate	No-Op
15 & 16	Exit Loop	No-Op	Exit Loop	No-Op	Terminate	No-Op
17	NA	No-Op	NA	No-Op	NA	No-Op
18, 1A & 1B	NA	NA	NA	NA	Terminate	No-Op
19	NA	NA	NA	NA	Terminate*	No-Op
1C & 1D	Exit Loop	String of all 1's	Exit Loop	No-Op	Terminate	No-Op
1E	Terminate*	No-Op	NA	NA	NA	NA
1F	Terminate	No-Op	NA	NA	NA	NA
28	NA	NA	NA	NA	Terminate*	No-Op
7D	Terminate data Transfer to Reg file.	No Data Transfer to Reg file	NA	NA	Terminate data Transfer from Reg file	No Data Transfer from Reg. file

* These instructions may terminate for reasons other than the exhausting of the field length.

NONTYPICAL (64-BIT FORMAT) INSTRUCTION TERMINATING CONDITIONS

Instruction Code	A Field {I}			B Field {I}			Z Field {O}		
	Result if A field is exhausted	Type of extension if any	A field length initially zero	Result if B field is exhausted	Type of extension if any	B field length initially zero	Result if Z field is exhausted	Z field length initially zero	Control Vector
B9	NA	NA	NA	NA	NA	NA	NA	NA	No
BB, BC & BD	NA	NA	NA	NA	NA	NA	Terminate {I}	No-Op {I}	No
C4, C5, C6 & C7	Extend	M-Zero	Extend	Extend	M-Zero	Extend	Terminate {O}	No-Op {O}	No
C8, C9 CA, CB	Terminate	NA	No-Op	Exit Search Iteration	NA	Exit Search Iteration	NA	NA	Yes(O)
CC	Terminate	NA	NO-OP	NA	NA	NA	NA	NA	No
CF	Terminate	NA	No-Op	Extend	M-Zero	Extend	NA	NA	No
D8 & D9	Terminate	NA	No-Op	NA	NA	NA	NA	NA	Yes (I)

3.1.4 Definitions and Rules

3.1.4.1 Overlap of Operand and Result Fields

If the result field overlaps a source field such that elements of the result are stored in the source field before elements in this portion of the source field are read, undefined results may occur. That is, the source elements may be the original elements or they may be the newly-stored elements. The instruction's results may become undefined. Note that some specific instructions prohibit any overlap of source and destination fields. This restriction is included in the appropriate instruction descriptions.

3.1.4.2 Self-Modifying Programs, Undefined Instructions and Undefined Operands

3.1.4.2.1 Self-Modifying Programs

As a general rule, self-modifying programs are not allowed. For further details and limitations see the individual model specification listed under section 2.0.

Programmer note: The 05 instruction "void stack and branch" should be used to ensure proper execution when utilizing self-modifying code.

3.1.4.2.2 Illegal Instructions

An instruction with an unused function code is termed an illegal instruction and causes the following:

A. If in Monitor Mode, an automatic branch to the address specified by the contents of absolute register 4 is executed.

B. If in Job Mode, an exchange to Monitor Mode is performed with execution beginning at the address specified by the contents of absolute register 3.

C. Any reference to the monitor or job's register file via an absolute or virtual bit address will be treated as if an illegal instruction had been performed.

3.1.4.2.3 Undefined Instructions

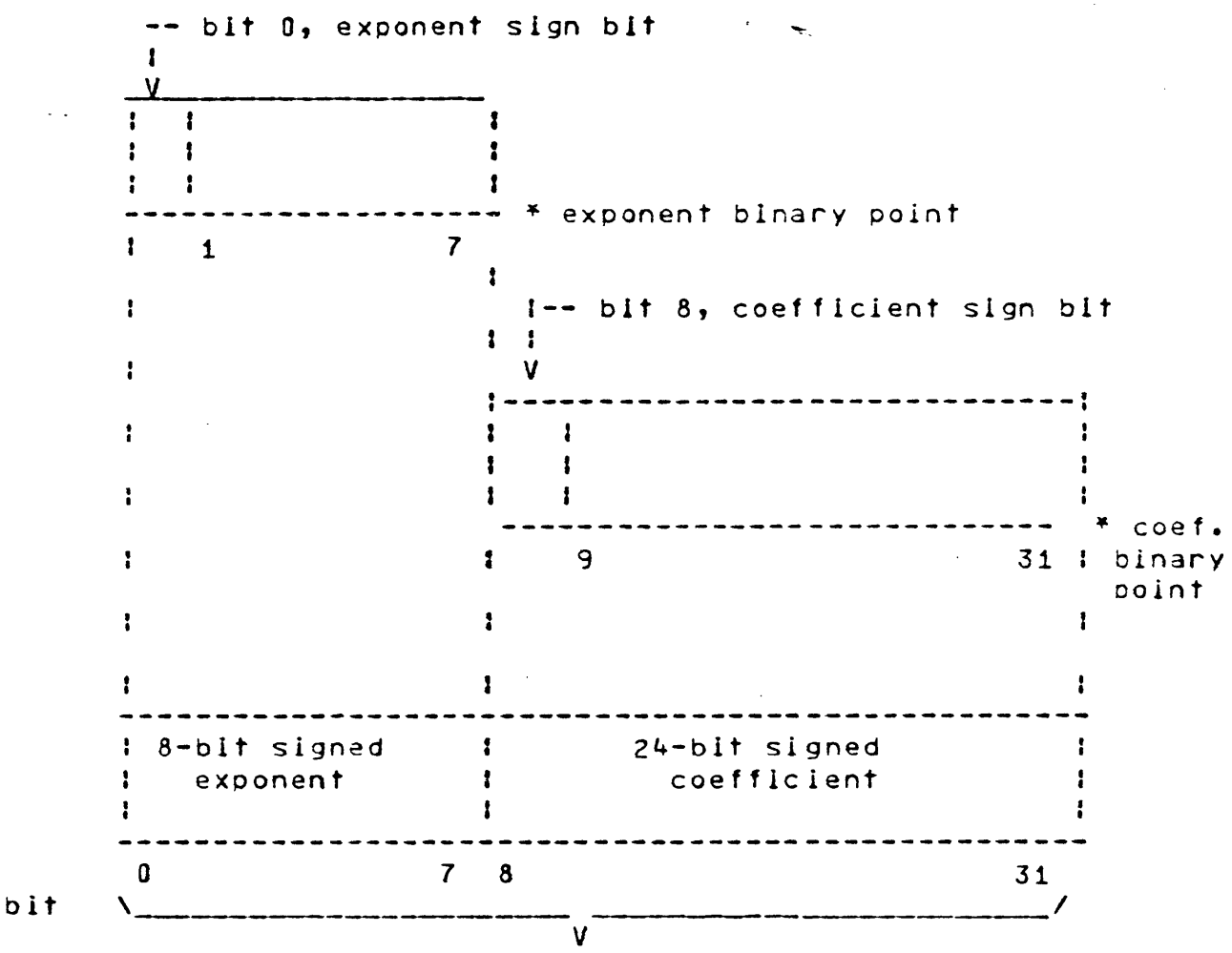
The instructions with a defined F code but which either have undefined bits set or specify an undefined operation cause undefined results. Note, that in Job Mode, the key-lock-virtual storage mechanism cannot be overcome even by an undefined instruction. Thus the only storage areas which can be affected are the pages assigned to the current job for which the write lockout bits are not set. Of course, in Monitor Mode no such memory protection exists.

3.1.4.2.5 No op Instructions

The instructions that are defined as No op instructions do not fetch data and do not alter data flags.

3.1.4.3 Floating Point Format

3.1.4.3.1 32-Bit Floating Point Format



32-bit floating point number

There are two 32-bit half-words in every 64-bit word. A 32-bit floating point number occupies a half-word.

A zero is a positive sign bit and a one is a negative sign bit for both the exponent and the coefficient.

Both the exponent and the coefficient are expressed as two's complement signed integers. Numbers are of the form (2**X)*c where c is the 24-bit signed coefficient, X is the 8-bit signed exponent, and the base is 2.

The range of coefficients is from 800000 to 7FFFFF base 16 which is from minus 8,388,608 to plus 8,388,607 base 10.

(Continued)

3.1.4.3.1 (cont'd)

The range of useful exponents is from 90 to 6F base 16 which is from minus 112 to plus 111 base 10. The values of 70 through 8F base 16 all fall into a special end case range as defined by the following table. X is any hexadecimal digit.

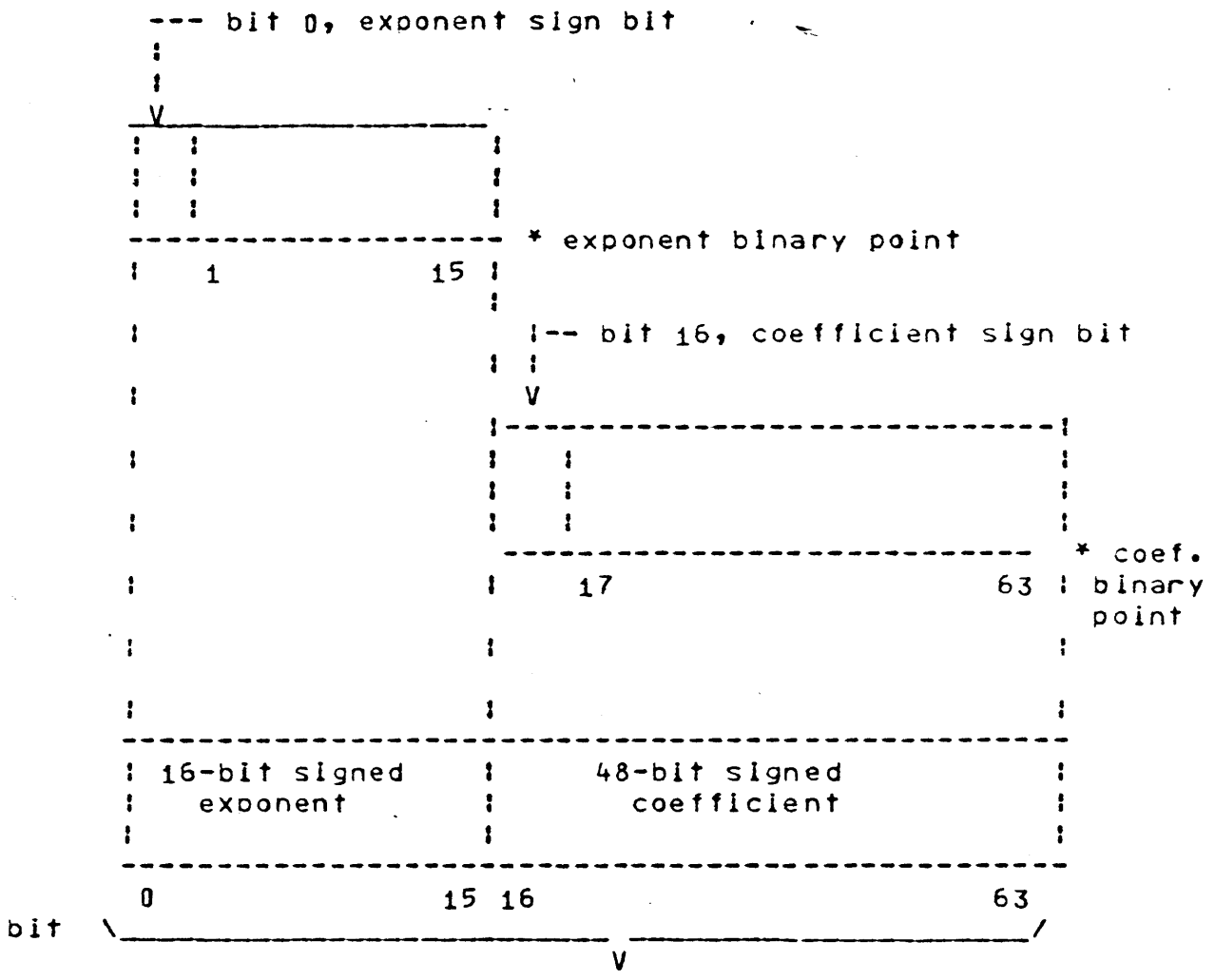
Element	Representation
Machine Zero	8XXXXXXX(H)
Indefinite	7XXXXXXX(H)

Examples of 32-bit floating point format represented in base 16.

+1	00	000001
+1 normalized	EA	400000
-1	00	FFFFFF
-1 normalized	E9	800000
+256[0]	00	000100

A floating point number is normalized if the coefficient sign bit is different from the next bit to the right. This condition implies that the coefficient has been shifted to the left as far as possible. Note that an all zero coefficient requires special attention for normalized operations (see 3.1.4.7).

3.1.4.3.2 64-bit Floating Point Format



64-bit floating point number

A 64-bit floating point number is contained in a 64-bit word.

A zero is a positive sign bit and a one is a negative sign bit for both the exponent and the coefficient.

Both the exponent and the coefficient are expressed as two's complement signed integers. Numbers are of the form (2**X)*c where c is the 48-bit signed coefficient, X is the 16-bit signed exponent, and the base is 2.

(Continued)

3.1.4.3.2 (cont'd)

The range of useful coefficients is from 8000 0000 0000 to 7FFF FFFF FFFF base 16 which is minus 140,737,488,355,328 to plus 140,737,488,355,327 base 10.

The range of useful exponents is from 9000 to 6FFF base 16 which is from minus 28,672 to plus 28,671 base 10. The values of 7000 through 8FFF base 16 all fall into a special end case range as defined by the following table. X is any hexadecimal digit.

Element	Representation
Machine Zero	8XXXXXXXXXXXXXXX(H)
Indefinite	7XXXXXXXXXXXXXXX(H)

Examples of floating point format represented in base 16

+1	0000 0000 0000 0001
+1 normalized	FFD2 4000 0000 0000
-1	0000 FFFF FFFF FFFF
-1 normalized	FFD1 8000 0000 0000
+256[D]	0000 0000 0000 0100

A floating point number is normalized if the coefficient sign bit is different from the next bit to the right. This condition implies that the coefficient has been shifted to the left as far as possible. Note that an all zero coefficient requires special attention for normalized operations (see 3.1.4.7).

3.1.4.4 End Cases

If indefinite is used as an operand in a floating point instruction, both the upper and the lower results are indefinite.

For the cases listed below, 0 represents machine zero and N represents an operand which is neither machine zero nor indefinite.

$0 \pm 0 = 0$	$0 \circ 0 = 0$	$0 / 0 =$ Indefinite
$0 \pm N = \pm N$	$0 \circ N = 0$	$0 / N = 0$
$N \pm 0 = N$	$N \circ 0 = 0$	$N / 0 =$ Indefinite

3.1.4.5 Floating Point Compare Rules

Several of the instructions compare two floating point operands for:

a. equality $(r) = (s)$
b. non-equality $(r) \neq (s)$
c. greater than or equal to $(r) \geq (s)$
d. less than $(r) < (s)$

For these examples, the first operand is represented by (r) and the second operand by (s).

3.1.4.5.1 One or Both Operands Indefinite

If one operand is indefinite, no compare condition is met since indefinite is not: greater than, less than, equal to, nor not equal to any other operand.

If both operands are indefinite, the $(r) = (s)$ and the $(r) \geq (s)$ conditions are met since indefinite is defined equal to indefinite.

3.1.4.5.2 Neither Operand Indefinite but One or Both Operands Machine Zero

Any non-indefinite, non-machine zero operand with a positive, non-zero, coefficient is strictly greater than machine zero.

Any non-indefinite, non-machine zero operand with a negative coefficient is strictly less than machine zero.

(continued)

3.1.4.5.2 (Cont'd)

Machine zero is equal only to itself and any number having a non-indefinite exponent and an all zero coefficient.

3.1.4.5.3 Neither Operand Indefinite Nor Machine Zero

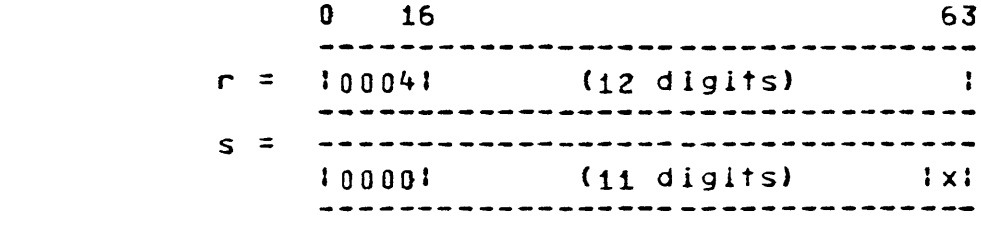
A. If the signs of the coefficients of the two operands are unlike, the operands are unequal and the operand with the positive coefficient is the larger of the two.
B. If the signs of the two coefficients are alike, a floating point subtract upper is performed; operand r minus operand s.

Condition met criteria are analyzed as follows for 64/32 bit compares:

a. If the upper 48/24 bits of the result coefficient are all zeros (r) = (s)
b. If the upper 48/24 bits of the result coefficient are not all zeros (r) ≠ (s)
c. If the result coefficient is positive (r) ≥ (s)
d. If the result coefficient is negative (r) < (s)

The above criteria (a and b) for equality and non-equality do not guarantee that if r = s, then s = r when the following is true:

a. The operands have unequal exponents.
b. "1" bits exist in any of the right-most bit positions of the coefficient which will be shifted off the right during alignment of the smaller exponent. For example:



Exponent difference = 4
If x = 0 then r = s implies s = r

(continued)

3.1.4.5.3 (Cont'd)

If $x \neq 0$ then if $r = s$, $-s \neq r$
or if $s = r$, $r \neq s$

The order of events of the floating point subtract upper is first to complement the subtrahend, then align the coefficient associated with the smaller exponent and finally to perform a floating point add operation. The following is an example of $r = s$ but $s \neq r$.

Operand r =	0100	0000	0000	1001	
s =	0104	0000	0000	0100	
Complement s	0104	FFFF	FFFF	FF00	
Align r	0104	0000	0000	0100	1
Add aligned r and complemented s	0104	0000	0000	0000	1

Since the upper 48 bits of the result coefficient are all zeros, the pair of operands are considered equal. However, if the operands are interchanged, the following happens:

Operand r =	0104	0000	0000	0100	
s =	0100	0000	0000	1001	
Complement s	0100	FFFF	FFFF	EFFF	
Align s	0104	FFFF	FFFF	FEFF	F
Add r and complemented, aligned s	0104	0000	0000	0100	
	0104	FFFF	FFFF	FEFF	F
	0104	FFFF	FFFF	FFFF	F

Since the upper 48 bits of the result coefficient are not all zeros, the pair of operands are considered unequal.

3.1.4.6 Upper and Lower Results

The floating point add, subtract and multiply instructions generate a result coefficient twice the length of the source operands' coefficients. The left and right halves of this result are called the upper result (U) and the lower result (L), respectively.

(continued)

3.1.4.6 (Cont'd)

The sign bit of the lower result's coefficient is not affected in a lower operation and remains at zero in two's complement arithmetic. The other bits of the lower coefficient receive no special treatment. Remember that a lower result is not meaningful alone, but it must be used in conjunction with its associated upper result. The data flags resulting from the lower result pertain only to the lower result.

3.1.4.6.1 Right Normalization

When the result coefficient overflows its register, a right shift of one place is necessary. In this case, the entire 95-bit (47-bit for 32 bit operands) result is shifted right one place with sign extension and one is added to the exponent. This operation is known as right-normalization and it is done, when necessary, even if normalization is not explicitly specified by the instruction. This may cause exponent overflow; if so, the result is set to indefinite and data flag bit 42 may be set.

3.1.4.6.2 Floating Point Add

Regardless of their signs, both operands' coefficients are extended to 94 bits (46 bits for 32 bit operands) in length, not including sign, by adding 47 (23 for 32 bit operands) zeros to the right of their binary points.

The exponents of the two operands are compared and the 94-bit (46-bit for 32 bit operands) coefficient of the operand having the smaller exponent is effectively shifted right one bit and its exponent increased by one, successively until the two exponents are equal. The sign of the shifted coefficient is extended from the left to the right during the shift. Negative coefficients approach a minus one and positive coefficients approach zero as they are shifted.

The add is a 94-bit (46-bit for 32 bit operands) operation, not including sign. Right normalization takes place, if necessary. The coefficient for the U result is the left-most 47 bits (23 bits for 32 bit operands) and the coefficient for the L result is the

(continued)

3.1.4.6.2 (Cont'd)

right-most 47 bits (25 bits for 32 bit operands) of the 94-bit (46-bit for 32 bit operands) result.

The exponent for the U result is equal to the larger of the two operand exponents. Right-normalization will increase this value by one, if it occurred.

The exponent for the L result is 47 (23 for 32 bit operands) base 10 less than the U result's exponent for all cases except three:

a. Right-normalization causes the U exponent to overflow; the U result is set to indefinite; the L exponent will be 6FD1 (59 in the 32-bit case) base 16.

b. If the U result's exponent minus 47 (23 for 32 bit operands) base 10 causes exponent underflow, machine zero is stored as the L result.

c. If either or both operands were indefinite, the U and L results are indefinite.

3.1.4.6.3 Floating Point Subtract

The floating point subtract operation is performed by complementing the coefficient of the subtrahend and performing a floating point addition operation. The complementation is a 48-bit (24-bit for 32 bit operands), two's complement operation and is performed before the operands are extended to 94 bits (46 bits for 32 bit operands).

Note that the subtract operation is not always commutative. In other words it is not always true that A-B = -(B-A). This characteristic will be observed if the following is true of A and B:

a. The exponents of A and B are not equal.

b. "1" bits exist in any of the right most bit positions of the coefficient which will be shifted off the right during alignment of the smaller exponent.

(continued)

3.1.4.6.3 (cont'd)

Example of A-B ≠ -(B-A):

```
 A =  0104   6FCB   807E   89F2
 B =  0100   6FAC   3F5D   A5FA <--
                                 :
                                 :
```

These two 1 bits will be shifted off during exponent alignment.

Complement B:

```
-B =  0100   9053   C0A2   5A06
```

Align B:

```
-B =  0104   F905   3C0A   25A0    6
```

A-B:

```
 A =  0104   6FCB   807E   89F2
-B =  0104   F905   3C0A   25A0    6
      0104   68D0   BC88   AF92    6
```

```
A-B = 0104   68D0   BC88   AF92
```

Align B:

```
 B =  0104   06FA   C3F5   DA5F    A
```

Complement A:

```
-A = 0104    9034   7F81   760E
```

-(B-A):

```
 B = 0104    06FA   C3F5   DA5F    A
-A = 0104    9034   7F81   760E
     0104    972F   4377   506D    A
-(B-A) = 0104   68D0   BC88   AF93
```

This differs from A-B in the last bit position.

3.1.4.6.4 Results of the Floating Point Multiply Instruction

When two floating point numbers are multiplied, the lower result retains the 47 (23 for 32 bit operands) least significant product bits generated. The sign bit of the lower result is always set to zero and the exponent of the lower result is the sum of the two source operands' exponents with the exceptions listed below:

The upper result retains the 47 (23 for 32 bit operands) product bits immediately to the left of the bits retained by the lower product. The sign of the upper product's coefficient follows the normal rules of algebra. The exponent of the upper result is the sum of the two source operands' exponents plus 47 (23 for 32 bit operands) with the following exceptions:

a. The sum of the source operands' exponents (plus 47 (23 for 32 bit operands) base 10, if upper result) exceed 6FFF (6F for 32 bit operands) base 16 for which case the result exponent is set to indefinite.

b. The sum of the source operands' exponents (plus 47 (23 for 32 bit operands) base 10, if upper result) is less than 9000 (90 for 32 bit operands) base 16 for which case the result exponent is set to machine zero.

c. Either or both operands are indefinite for which case the result exponent is set to indefinite.

d. Neither operand is indefinite but either or both operands are machine zero, for which case the result exponent is set to machine zero.

Except for the calculation of significance, if either operand has a coefficient of 8000 0000 0000 (800000 for 32 bit operands) base 16 and an exponent of X, the operand will be treated as though its coefficient were C000 0000 0000 (C00000 for 32 bit operands) base 16 and its exponent were X+1.

3.1.4.6.5 The Floating Point Divide Instruction

The quotient from the divide operation is the result of dividing the prenormalized, integer coefficient of the divisor into the integer coefficient of the dividend generating a 47-bit (23-bit for 32-bit operand quotients). Except for the calculation of

(Continued)

3.1.4.6.5 (cont'd)

significance, if either operand has a coefficient of 8000 0000 0000 (800000 for 32 bit operands) base 16, the operand will be handled as though its coefficient were C000 0000 0000 (C00000 for 32 bit operands) base 16 and its exponent increased by one. When the divide hardware normalizes the divisor coefficient, the number of places shifted left is added to the exponent of the quotient as defined below.

The exponent of the result will be given by the following equation:

Exponent of Quotient = (Exponent of Dividend)
- (Exponent of Divisor)
- (constant - NC)

where the constant is 46 (22 for 32 bit operands) base 10 and NC is the number of places shifted left to prenormalize the divisor.

The right-most bit of the quotient is neither rounded nor adjusted. The remainder is not retained. The sign of the quotient's coefficient follows the normal rules of algebra.

3.1.4.6.6 Normalized Upper Results

The normalized add and subtract instructions generate an intermediate result identical to the final result of the add U and the subtract U instructions. Normalization of the intermediate, 48-bit (24-bit for 32 bit operands) coefficient result then takes place as follows:

The coefficient is shifted left one bit and its exponent is decreased by one, successively, until the sign bit and the bit immediately to the right of the sign bit are different. During this shift, zeros are attached to the right end of the coefficient. If reducing the exponent by one causes exponent underflow, the result of the normalization operation is defined as machine zero.

3.1.4.6.7 Double Precision Results

Several instructions

DA SUM (A0 + A1 + A2 ... + AN) TO C AND C + 1
DC VECTOR DOT PRODUCT TO C AND C + 1

produce double precision results. The double precision add operation is nothing more than a floating point add producing both an upper and lower result simultaneously and retaining both of these results for the next floating point add operation. Thus the partial result in 64-bit arithmetic consists of 94 coefficient bits plus sign information and in 32-bit arithmetic consists of 46 bits plus sign information. The DOT PRODUCT instructions add both the upper and lower results of the multiply operations to the partial results of the add operations as described above.

Because of speed consideration, the accumulative results for double precision are order dependent and may vary from model to model. Precautions will be taken to insure that results do not vary on a particular model due to interrupts.

3.1.4.7 Floating Point Square Root

Except for the calculation of significance, if the operand has a coefficient of 8000 0000 0000 (800000 for 32 bit operands) base 16 the operand will be handled as though its coefficient were C000 0000 0000 (C00000 for 32 bit operands) base 16 and its exponent increased by one.

The result of a floating point square root operation is produced by performing the following steps:

1. Determine and record the significance of the coefficient of the input operand.

2. Transform the input operand into its positive form.
3. If the exponent of the input operand is odd, reduce it by one and multiply the coefficient from step 2 by two. If the exponent is even, do not modify it.

(continued)

3.1.4.7 (cont'd)

4. Obtain the square root of the coefficient from step 3. Attach enough "0" bits to the right end of the coefficient to allow 47 answer bits to be produced (23 answer bits for the 32-bit square root).

5. If the original input operand was negative, complement the 47 (23 for 32 bit operands) answer bits produced in step 4. If the original input operand was positive, do not modify the answer bits from step 4.

6. Form a result exponent by dividing the exponent from step 3 by two and subtracting 23 from it (subtract 11 for the 32-bit square root).

7. Adjust the answer bits from step 5 so that they produce a coefficient with the same significance as that recorded from the input operand in step 1. Adjust the exponent obtained in step 6 so as to compensate for the change in magnitude of the result coefficient.

8. A source operand having an all-zero coefficient will produce a result with an all-zero coefficient whose exponent has been effectively divided by two by being right shifted one place with sign extension. If the source operand is negative, data flag bit 45 is set. If the source operand is indefinite or machine zero, the result will be indefinite or machine zero, respectively. In these two cases, data flag bit 45 is not set.

3.1.4.8 Significant Results

The significant bit count for a floating point number is equal to the number of bit positions in the coefficient (excluding the sign bit) minus the left shift count necessary to normalize that number. An all zero (or an all one) coefficient has a significant bit count of zero. Note that for a non-zero coefficient that is an exact power of two, the positive form of the coefficient has a significant bit count that is one greater than the

(continued)

3.1.4.8 (Cont'd)

significant bit count of the negative form of the coefficient. The significance of an input operand is determined from the operand as originally read from a register or from central storage before any operations such as sign control, the handling of a coefficient of 8000 0000 0000 (800000 for 32 bit operands) or the left shift for odd exponents in square root are performed.

Significant arithmetic determines which of the source operands has the smaller significant bit count and records that count; and then, after the arithmetic operation, determines the significant bit count of the result after any necessary sign correction. The input significant bit count and the result significant bit count are then compared. If the result significant bit count is less than the input significant bit count, the result coefficient is left shifted (with zeros shifted in) by the difference in significant bit counts and the exponent is reduced accordingly. If the result and input significant bit counts are equal, the coefficient is not shifted nor the exponent adjusted. If the result significant bit count is greater than the input significant bit count, the result coefficient is right shifted (end-off with sign extenson) and the exponent increased accordingly. Note that for multiply, the entire 95 bit result (47 bits for 32-bit multiply) is shifted as required.

Exponent overflow, exponent underflow and divide fault cause forced results as usual. Adjusting for significance can cause exponent overflow or underflow or it can take a result out of exponent overflow or underflow.

3.1.4.9 Sign Control

Certain vector, sparse vector and non-typical instructions provide an operation called sign control on the input operands. For these insructions, bits 5, 6, and 7 of the G field have the following significance.

Bit 5	Bit 6	
0	0	Use the operands from the A stream in the normal manner.

(continued)

3.1.4.9 (Cont'd)

0	1	Complement the coefficients of the operands from the A stream before using them.
1	0	Use the magnitude of the coefficients of the operands from the A stream.
1	1	Complement all positive coefficients of the operands from the A stream before using them. Negative operands will not be altered.

Bit 7

0	Use the operands from the B stream in the normal manner.
1	Use the magnitude of the coefficients of the operands from the B stream.

Any complementation necessary to achieve the required operand state is a two's complement operation and is performed before operands are used in the specified arithmetic operation. Complementation in sign control is as described in section 3.1.4.6.3 "Floating Point Subtract".

Any significance calculation necessary in performing an instruction is made before the above mentioned complementation occurs.

(continued)

3.1.4.9 (Cont'd)

The following instructions have sign control:

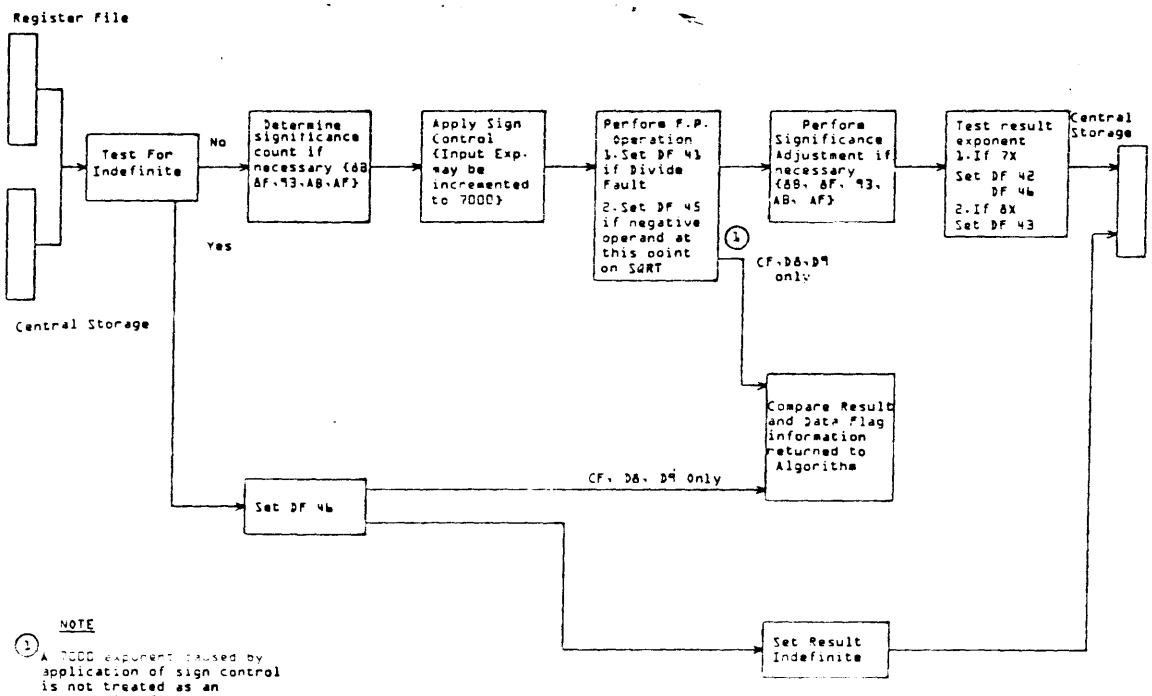
Instruction		A Operands (Bit 5 & Bit 6)		B Operands (Bit 7)
80,81,82	Vector Add	X	X	X
84,85,86	Vector Subtract	X	X	X
88,89,8B	Vector Multiply	X	X	X
8C,8F	Vector Divide	X	X	X
93	Vector Square Root	X	X	0
A0,A1,A2	Sparse Vector Add	X	X	X
A4,A5,A6	Sparse Vector Subtract	X	X	X
A8,A9,AB	Sparse Vector Multiply	X	X	X
AC,AF	Sparse Vector Divide	X	X	X
CF	Arithmetic Compress	X	X	X
D8	Maximum of A to C	X	0	0
D9	Minimum of A to C	X	0	0

X - 0 or 1 bit is legal

0 - This bit must always be set to zero

The Operand Flow Chart on the following page illustrates the order of operations when sign control is selected.

OPERAND FLOW FOR INSTRUCTIONS HAVING SIGN CONTROL



NOTE

(1) A 7000 exponent caused by application of sign control is not treated as an operand indefinite by the floating point compare.

3.1.5 Item Count (field lengths, offsets, indices, etc.)

All field lengths, offsets, indices, shift counts, etc., are item counts which specify a number of bits, bytes, half-words or words.

String Indices

In all string instructions indices are item counts in bytes. Indices are different from the offsets in the vector instructions. Offsets are limited to (2**16) -1 while string indices are limited to (2**45) -1 for byte item counts and (2**42) -1 for word item counts). Since byte indices are left-shifted three places before they are added to a base address, the left-most three bits of a string index are not used and must consist of extended sign. In a similar manner the left-most six bits of word index must consist of extended sign. Overflows are ignored when adding indices to base addresses.

Where an item count other than an index is contained in a 48-bit field, there shall be at least 32 consecutive and identical sign bits. Sign bits must always be extended to the left to fill the 16-bit or 48-bit field containing it.
The item count unit is specified by the instruction title line code (see arrow).

Example

```
                              :
                              V
3.2.1.249        F8   3   8   ST MOVE BYTES LEFT; A->C
```

The 8 indicates that field lengths and indices are expressed in bytes. Any deviation from this method of specifying the units for the various item counts would be indicated in the instruction description or in the description of the instruction type. The instruction type refers to ST (string), VT (vector), etc.

An index may be either positive or negative in sign. The maximum magnitude of an index is a function of its usage. The index is shifted to the left end-off

(continued)

3.1.5 (Cont'd)

zero/three/five/six places before the addition to the base address when the unit for the index is bits/bytes/half-words/words.

An offset may be either positive or negative in sign and must have a magnitude of less than (2**16).

A field length must be positive in sign and have a magnitude of less than (2**16); the use of a negative field length causes that length to become strictly undefined. Offsets are subtracted from the field length in vector instruction, but note that for a negative offset, this amounts to increasing the length specification since subtracting a negative quantity is addition.

3.1.6 Data Flag Branch Register

3.1.6.1 General Description

The data flag register is designed to give the programmer an automatic branch to a special routine for certain operands, results, conditions, etc., without his having to pay the time penalty of explicitly checking these conditions in his program. If a condition which has been previously selected to cause an automatic branch occurs during an instruction, the address of the next instruction which would have been executed is stored into the address portion of register 01 and a branch is made to the address contained in register 02. Zero, one or several more instructions may be executed before an automatic branch actually takes place. The amount varies from model to model.

The data flag register is stored into word four of the invisible package.

3.1.6.2 Register Description

PRODUCT FIELD	MASK FIELD	DATA FLAGS	FREE FLAGS
16 bits	16 bits	16 bits	16 bits
* 3 15	* 19 31	* 35 47	* 51 63
0 2	16 18	32 34	48 50 63

*Bits 0 through 2, 16 through 18, 32 through 34, and 48 through 50 of the data flag register are undefined. Any attempt to sample, set or clear these bits is meaningless and the result of any instruction trying to do so is undefined.

3.1.6.2.1 Data Flag Bits

Data flags 35-47 indicate conditions that have occurred; i.e, bit 37 is set at the end of a CC instruction if no match is found. Note that another CC instruction which finds a match will not clear bit 37. Bits 35-47 are cleared only by the Data Flag Register Bit Branch and Alter, and the Data Flag Register Load/Store instructions.

For data flags 41 through 46, inclusive, if a control vector is being used, the current control vector bit must be permissive in order to set any of the data flags; i.e., if a divide fault occurred, but the control vector bit for that result element was not permissive, the divide fault data flag would not be set by that result element.

3.1.6.2.2 Mask Bits

A mask bit is associated with each of the data flags. The mask bits have the function of selecting the conditions for which the programmer wishes an automatic data flag branch.

(continued)

3.1.6.2.2 (Cont'd)

It is important to note that the associated mask bit need NOT be set in order to set a data flag bit. The mask function is solely one of enabling a particular data flag to cause a bit to set in the product field. The order in which the mask bit and its associated data flag bit are set is immaterial, as the result is the same; that is, their associated product bit is set.

3.1.6.2.3 Product Bits

Each product bit is the dynamic logical product of a data flag bit and its associated mask bit. Data flag branches are performed when there is at least one <u>one</u> in the product register and the data flag branch enable bit is set.

3.1.6.2.4 Data Flag Branch Enable Bit

The data flag branch enable bit, bit 52, must be set for an automatic data flag branch (DFB) to occur. Bit 52 is automatically cleared by the hardware when a DFB takes place. It must be reset with a Data Flag Register Bit Branch and Alter or a Data Flag Register Load/Store Instruction to re-enable the DFB.

3.1.6.2.5 Data Flag Register Bit Assignments

```
Product Bit
!
!  Mask Bit
!  !
!  ! -Data Bit
!  ! !
V  V V
3-19-35
```

Soft Interrupt. Monitor software can set bit 35 of a job's Data Flag Branch register while the register is stored in the job's Invisible Package. If, after exchanging back to job mode, bit 35 and its corresponding mask bit (bit 19) are set, a normal Data Flag Branch occurs.

(continued)

3.1.6.2.5 (Cont'd)

4-20-36

Job Interval Timer

5-21-37

Select, condition not met. Instructions C0 through C3. No match on CC instruction.

6-22-38

Unused.

7-23-39

The binary result exceeds the range of +((2**47)-1) to -(2**47) for the 10 instruction.

8-24-40

Bit 40 is the inclusive OR of bits 37, 38 and 39. Bit 24 masks bit 40. Bit 8 is the logical product of bits 24 and 40.

9-25-41

Floating point divide fault: The divisor has an all zero coefficient or the divisor as read from the register file or from central storage is machine zero. If the divisor and/or the dividend is indefinite, no divide fault exists. If a divisor causes a divide fault, the quotient is set to indefinite. The exponent overflow and result machine zero data faults are not set by a divide whose divisor caused a divide fault.

10-26-42

Exponent overflow: The exponent of the result is larger than 6FFF (6F for 32-bit arithmetic) base 16. Results are not checked for exponent overflow until after the exponent adjustment for normalization or significance has taken place. In the adjust exponent instructions, if a left shift exceeds the number of

(continued)

3.1.6.2.5 (Cont'd)

places required for normalization, this data flag is set. Exponent overflow causes the result to be set to indefinite; therefore, the indefinite flag will

always be set on a exponent overflow. This exponent overflow data flag is not set if either source operand from central storage or the register file is indefinite or by a divide instruction whose divisor causes a divide fault.

11-27-43

Result Machine Zero: The exponent of the result returned to central storage or to the register file is less than 9000 (90 for 32-bit arithmetic) base 16. Result Machine Zero may be caused by exponent underflow or by one or more of the input operands being machine zero. The Result Machine Zero data flag bit is not set by a divide whose divisor causes a divide fault.

12-28-44

Bit 44 is the inclusive OR of bits 41, 42 and 43. Bit 28 masks bit 44. Bit 12 is the logical product of bits 28 and 44.

13-29-45

A negative source operand was encountered in a square root instruction. The square root of the absolute value of the operand is formed; and the two's complement of this square root is stored as the result.

14-30-46

An indefinite result was placed into central storage or into the register file.... or .... either or both operands of a floating point compare were indefinite.

An indefinite result may be caused by one or both operands of a floating point arithmetic operaton being indefinite or by the occurrence of either a divide fault or an exponent overflow.

15-31-47

A breakpoint bit has occurred. (See the 04 Instruction description).

3.1.6.2.6 Free Data Flags

Bit 51 is the dynamic inclusive OR of the product field. This bit is set if any of bits 4 through 15 are set. Bit 51 cannot be cleared directly.

Bit 52 is the data flag branch enable bit. If bit 52 is a one and bit 51 becomes a one (or vice versa) a data flag branch occurs at the end of the current instruction. See 3.1.6.3 for additional information. Bit 52 is automatically cleared by the execution of a data flag branch.

Bits 53, 54 and 55
There are no product or mask bits associated with bits 53, 54, and 55. Bits 53, 54, and 55 are initialized during the initial phases of the instructions (unless the instruction is a no op -- see Section 3.1.3) which may set any of them. Thus, if pertinent, these bits must be sampled before executing another instruction which would alter their previous state. The setting of bits 53, 54, and 55 does not cause a data flag branch.

Bits 56 through 63
There are no product or mask bits associated with bits 56 through 63. The purpose of bits 58 through 63 is to assist software in determining what operation caused data flag bits 41, 42, 43, 45 and 46 to go set. For instance, if after an automatic data flag branch bit 58 was set it would indicate that the operation causing the fault was issued early in the program because of the execution time of divide/square root is much longer than say a multiply or add. Because of possible parallel operation between scalar and vector bits 59 through 63 being set would indicate that bit 41, 42, 43, 45 and/or 46 was set by the vector operation but could have also been set by a scalar operation. If bits 59 through 63 are not set and yet 41, 42, 43, 45 and/or 46 are set indicates that they were set only by a scalar operation. Example, if a vector operation generates a divide fault

(continued)

3.1.6.2.6 (Cont'd)

both bits 41 and 59 will set as a divide fault but a scalar instruction alone will only set bit 41.

Bit 53 - Result field all zeros
Bit 54 - Result field mixed
Bit 55 - Result field all ones
→ logical string instructions F0 through F7

Bit 53 - Ones were counted
Bit 54 - Undefined
Bit 55 - Undefined
→ 1E count leading equals

Bit 53 - Undefined
Bit 54 - Multiple hits
Bit 55 - Undefined
→ 08 and 09 Maximum and minimum

Bit 53 - Whole field scan, no hit
Bit 54 - Undefined
Bit 55 - Undefined
→ 28, scan equal

Bit 56 - A CPU gate associated with the Maintenance Station monitoring counters (See Functional Computer Specification listed in Section 2.0)

Bit 57 - A CPU gate associated with the Maintenance Station monitoring counters (See Functional Computer Specification listed in Section 2.0)

Bit 58 - A scalar divide/square root operation set bits 41, 42, 43, 45 and/or 46.

Bit 59 - Vector box floating point divide fault, duplicate of bit 41 caused by a vector.

Bit 60 - Vector box Exponent overflow, duplicate of bit 42 caused by a vector.

Bit 61 - Vector box machine zero result, duplicate of bit 43 caused by a vector.

(continued)

3.1.6.2.6 (Cont'd)

Bit 62 - Vector box square root result imaginary, duplicate of bit 45 caused by a vector.

Bit 63 - Vector box indefinite result, duplicate of bit 46 caused by a vector.

OP CODE V	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
00										
01										
02										
03										
04									X	
05										
06										
07										
08										
09										
0A										
0B										
0C										
0D										
0E										
0F										
10			X							
11										
12										
13										
14										
15										
16										
17										
18										
19										
1A										
1B										
1C										
1D										
1E										X
1F										
20								X		
21								X		
22								X		
23								X		
24								X		
25								X		
26								X		
27								X		
28										X
29										
2A										
2B										
2C										
2D										
2E										
2F										
30										
31										
32										
33										
34										
35										
36										
37										
38										
39										
3A										
3B										
3C										
3D										
3E										
3F										

OP CODE V	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
40				X	X		X			
41					X		X			
42				X	X		X			
43										
44				X	X		X			
45					X		X			
46				X	X		X			
47										
48				X	X		X			
49				X	X		X			
4A										
4B				X	X		X			
4C			X	X	X		X			
4D										
4E										
4F			X	X	X		X			
50							X			
51							X			
52							X			
53					X	X	X			
54				X	X		X			
55				X			X			
56										
57										
58										
59				X	X		X			
5A										
5B										
5C					X		X			
5D					X		X			
5E										
5F										

OP CODE V	DATA FLAG BITS 37	38	39	41	42	43	45	46	47	53 54 55
60				X	X		X			
61					X		X			
62				X	X		X			
63										
64				X	X		X			
65					X		X			
66				X	X		X			
67										
68				X	X		X			
69				X	X		X			
6A										
6B				X	X		X			
6C			X	X	X		X			
6D										
6E										
6F			X	X	X		X			
70							X			
71							X			
72							X			
73					X	X	X			
74				X	X		X			
75				X			X			
76				X	X		X			
77				X	X		X			
78										
79				X	X		X			
7A										
7B										
7C										
7D										
7E										
7F										

SUPER COMPUTER OPERATIONS

OP COD	DATA FLAG BITS V	37	38	39	41	42	43	45	46	47	53 54 55	OP CODE	DATA FLAG BITS V	37	38	39	41	42	43	45	46	47	53 54 55
80					X	X		X				A0					X	X		X			
81						X		X				A1					X	X		X			
82					X	X		X				A2					X	X		X			
83												A3											
84					X	X		X				A4					X	X		X			
85						X		X				A5					X	X		X			
86					X	X		X				A6					X	X		X			
87												A7											
88					X	X		X				A8					X	X		X			
89					X	X		X				A9					X	X		X			
8A												AA											
8B					X	X		X				AB					X	X		X			
8C				X	X	X		X				AC				X	X	X		X			
8D												AD											
8E												AE											
8F				X	X	X		X				AF				X	X	X		X			
90								X				B0											
91								X				B1											
92								X				B2											
93						X	X	X				B3											
94					X	X		X				B4											
95					X			X				B5											
96					X	X		X				B6											
97					X	X		X				B7											
98												B8											
99					X	X		X				B9											
9A												BA											
9B												BB											
9C						X		X				BC											
9D												BD											
9E												BE											
9F												BF											

------- S U P E R C O M P U T E R O P E R A T I O N S --------

OP CODE	DATA FLAG BITS V	37	38	39	41	42	43	45	46	47	53 54 55	OP CODE	DATA FLAG BITS V	37	38	39	41	42	43	45	46	47	53 54 55
C0	X							X				E0											
C1	X							X				E1											
C2	X							X				E2											
C3	X							X				E3											
C4								X				E4											
C5								X				E5											
C6								X				E6											
C7								X				E7											
C8								X				E8											
C9								X				E9											
CA								X				EA											
CB								X				EB											
CC	X											EC											
CD												ED											
CE												EE											
CF								X				EF											
D0						X		X				F0										X	
D1						X		X				F1										X	
D2												F2										X	
D3												F3										X	
D4						X		X				F4										X	
D5					X	X		X				F5										X	
D6												F6										X	
D7												F7										X	
D8								X		X		F8											
D9								X		X		F9											
DA					X	X		X				FA											
DB					X	X		X				FB											
DC					X	X		X				FC											
DD												FD											
DE												FE											
DF					X	X		X				FF											

3.1.6.3 Data Flag Branch (DFB)

If a bit in the mask field is set and its associated masked data flag bit is set, the associated bit in the product field becomes a one. Bit 51 in the free flag field also becomes a one since it is the dynamic inclusive OR of bits 4 through 15 of the product field.

If bit 51 is a one from above and if bit 52 is also set (this is the DFB enable bit), an automatic DFB occurs. The DFB takes place sometime following the termination of the instruction which caused the DFB condition to exist. The execution of the DFB sets the bit address of the next instruction into the right-most 48 bits of register 01 and a branch is made to the bit address contained in the right-most 48 bits of register 02. The DFB enable bit in the flag mask register (bit 52) is automatically cleared at this time. The left-most 16 bits of register 01 are cleared to zero by a DFB. The address in register 01 points to zero, one or more instructions removed from the instruction causing the DFB.

Programmer Note:

DFB's are disabled when bit 52 is cleared. But if bit 52 is reset before eliminating all the DFB conditions, another DFB will occur which will change the return address in register 01 and the machine may wind up in a "tight loop" if proper caution is not taken. Sampling bit 51 for a zero before setting bit 52 will prevent this situation for all cases except those involving the Job Interval Timer. When using the Job Interval Timer, it should be remembered that the setting of bit 36 in the DFR occurs asynchronously with respect to instruction execution once the Job Interval Timer is loaded. Thus the timer may set bit 36 after the check of bit 51 and before the branch to the contents of register 01. One method of handling this situation is to examine the contents of register 01 upon entering the routine for handling Data Flag Branches. If register 01 indicates that the branch occurred outside the DFB routine, then register 01 could be copied to a temporary location.

(continued)

3.1.6.3 (cont'd)

If register 01 indicated that the branch had occurred within the DFB routine, then register 01 would not be copied to the temporary location. At the conclusion of the DFB routine, a branch would always be taken to the contents of the temporary location.

A simpler method is to combine the setting of bit 52 and the branch to the contents of register 01 into a single 33 instruction (33603401).

3.1.7 Register File

For register operations, the 8-bit instruction designators directly address the 256[D] registers of the register file. During program execution (monitor or job), these registers reside in the CPU's register file. When an exchange operation occurs, the registers are stored into the first 256[D] memory locations of the particular job or monitor mode program beginning at bit address zero (absolute address if in monitor mode and virtual address if in job mode). The registers may not be referenced as memory by their associated monitor or job program. The only exceptions to this rule are the B7 and BA instructions with G-bit 7 set. (See the B7 and BA instructions in this specification).

Figure 1 shows a map of the register file and the relationship between the register, its storage address and its 8-bit designator. The number on the right represents the bit address and the number on the left is the value of the 8-bit designator for the 64-bit register case. The number inside the register represents the value of the 8-bit designator for the 32-bit operand case. Note that any reference to 32-bit register one is undefined.

(continued)

3.1.7 (cont'd)

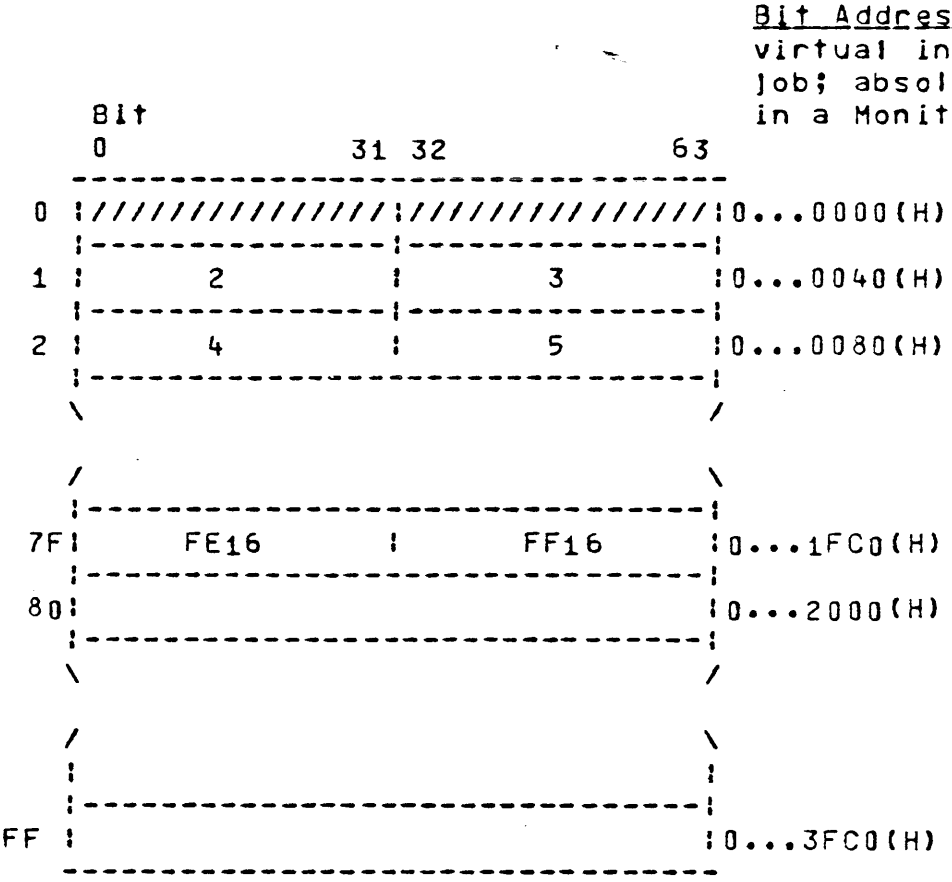


Figure 1 Register File

Register File Restrictions

A. Register Zero (Job or Monitor Mode)

1. During an exchange operation the contents of the trace register (register zero) and the appropriate memory location for register zero are exchanged (swapped).

Monitor to Job:

	Before Exchange	After Exchange
Absolute Address Zero	A	C
Virtual Address Zero	B	B
Trace Register	C	B

(continued)

3.1.7 (cont'd)

Job to Monitor:

	Before Exchange	After Exchange
Absolute Address Zero	A	A
Virtual Address Zero	B	C
Trace Register	C	A

If monitor and job mode share a common register file (see E this section), the following will occur upon an exchange operation (monitor to job or job to monitor):

	Before Exchange	After Exchange
Absolute Address Zero	A	B
Virtual Address Zero	A	B
Trace Register	B	B

During a 7D (Swap) instruction involving register zero as part of the register field, note a required peculiarity. Although the current contents of the trace register are sent to the appropriate memory location for register zero, the current contents of the trace register are not altered.

	Contents Before 7D	Contents After 7D
Memory location for register zero	A	B
Trace register	B	B

(continued)

3.1.7 (cont'd)

2. Register Zero when referenced by a designator will provide machine zero as an operand except when used as a source register for a base address or other description for a vector or string instruction, in which case register zero will appear to contain 64-zero bits. The use of a zero address may cause the instruction to be treated as an illegal instruction as defined in Section 3.1.10. If register zero is specified as the destination register, the instruction typically performs normally with data flags being set, if warranted, but no data is stored. Some instructions become undefined if register zero is specified as a destination register.

The table 3.1.7-1 is intended to define what operand is obtained when register zero is specified for a source operand. To simplify this chart, the use of register zero as a destination register has been ignored. A blank in the chart indicates where it is either not possible to specify register zero or it may only be specified as a destination register. The designators R, S, T, G, X, A, Y, B, Z and C are used for convenience although they do not apply to all instructions. Utilization of the following symbols is made.

Symbol	Result When Register Zero is Referenced for an Operand
M	Machine zero is provided. 8000 0000 0000 0000[H] 64-bit mode 8000 0000[H] 32-bit mode
A	All zero is provided.
Z	All zero in the used portion. In this instance the left-most bit is not used thus machine zero and all zeros are indistinguishable.
N	Instruction performs as a no-op.

(continued)

3.1.7 (cont'd)

C — No control vector is used.

O — A mask of all ones is provided.

3. The BA instruction can read register zero for data. See the respective instruction descriptions for details.

B. 64-Bit Registers One and Two (32-bit registers 2 through 5)

If data flag branches are being used, 64-bit registers one and two must be reserved exclusively for that use. Register one is the data flag branch exit address and register two holds the data flag branch entry address.

C. Monitor's 64-Bit Registers 0-F[H] (32-bit registers 0-1F[H]

Registers zero, one and two have the restrictions listed in A and B above. Registers 3 through 7 are used for the illegal instruction, exit force, external interrupt and storage access interrupt entry points.

D. 32-Bit Register One (right-most half of 64-bit register 0)

Any reference to 32-bit register one is undefined.

E. Common Register File for Monitor and Job Modes

Monitor and job modes will have perfectly overlapping register files if monitor executes an Exit Force instruction (09) with either designator S or the contents of register S equal to zero. In an exchange from monitor to job mode, the monitor's register file is stored starting at absolute bit address zero, while the job's register file is loaded from the first 256 locations of its virtual page zero. Register S contains the absolute address of the jobs virtual page zero (see Exit Force instruction). When register S specifies an address of zero, the

(continued)

3.1.7 (cont'd)

register file contents are not changed. This applies to exchange in both directions. Also, since the right-most 15 bits of register S must contain zeros (see Exit Force instruction), only a perfect overlap may occur.

Table 3.1.7-1

Op Code	Instruction Designator R	S	T
00			
03			
04	Z		
05			Z
06			
08			
09		Z	Z
0A	Z		
0C			
0D			
0E	Z	Z	
0F	Z	A	
10	M		
11	Z		
12	Z	Z	
13	Z	Z	Z
14	A	A	A
15	A	A	A
16	A	A	A
17			
18			
19			
1A			
1B			
1C	A	A	A
1D	A	A	A
1E	A	Z	
1F	A	Z	

Op Code	Instruction Designator R	S	T
20	M	M	Z
21	M	M	Z
22	M	M	Z
23	M	M	Z
24	M	M	Z
25	M	M	Z
26	M	M	Z
27	M	M	Z
28		Z	A
29			
2A			
2B	M	Z	
2C	M	M	
2D	M	M	
2E	M	M	
2F		Z	Z
30	M		
31	Z	Z	Z
32		Z	Z
33			Z
34	M	Z	
35	Z	Z	Z
36		Z	Z
37			
38	M		
39			
3A	Z		
3B	Z		
3C	Z	Z	
3D	Z	Z	
3E			
3F	Z		

Table 3.1.7-1

Op Code	Instruction Designators R	S	T	Op Code	Instruction Designators R	S	T
40	M	M		60	M	M	
41	M	M		61	M	M	
42	M	M		62	M	M	
				63	M	Z	
44	M	M		64	M	M	
45	M	M		65	M	M	
46	M	M		66	M	M	
				67	M	Z	
48	M	M		68	M	M	
49	M	M		69	M	M	
4B	M	M		6B	M	M	
4C	M	M		6C	M	M	
4D				6D	M	Z	
4E	Z			6E	M	Z	
4F	M	M		6F	M	M	
50	M			70	M		
51	M			71	M		
52	M			72	M		
53	M			73	M		
54	M	Z		74	M	Z	
55	M	M		75	M	Z	
56				76	M		
				77	M		
58	M			78	M		
59	M			79	M		
5A	M			7A	M		
5B	Z	Z		7B	Z	Z	
5C	M			7C	M		
5D	M			7D	A	*	A
5E	Z	Z		7E	Z	Z	
5F	Z	Z	M	7F	Z	Z	M

*See Section 3.1.7 paragraph A.1

Table 3.1.7-1

Op Code	Instruction Designators G	X	A	Y	B	Z	C
80		Z	A*	Z	A*	C	A
81		Z	A*	Z	A*	C	A
82		Z	A*	Z	A*	C	A
83		Z	A*	Z	A*	C	A
84		Z	A*	Z	A*	C	A
85		Z	A*	Z	A*	C	A
86		Z	A*	Z	A*	C	A
86		Z	A*	Z	A*	C	A
88		Z	A*	Z	A*	C	A
89		Z	A*	Z	A*	C	A
8A		Z	A*	Z	A*	C	A
8B		Z	A*	Z	A*	C	A
8C		Z	A*	Z	A*	C	A
8F		Z	A*	Z	A*	C	A
90		Z	A*			C	A
91		Z	A*			C	A
91		Z	A*			C	A
93		Z	A*			C	A
94		Z	A*	Z	A*	C	A
95		Z	A*	Z	A*	C	A
96		Z	A*			C	A
97		Z	A*			C	A
98		Z	A*			C	A
99		Z	A*			C	A
9A		Z	A*			C	A
9B		Z	A*	Z	A*	C	A
9C		Z	A*			C	A
90		Z	A*	Z	A*	C	A

Op Code	Instruction Designators G	X	A	Y	B	Z	C
A0		A	Z*	A	Z*	A	Z
A1		A	Z*	A	Z*	A	Z
A2		A	Z*	A	Z*	A	Z
A4		A	Z*	A	Z*	A	Z
A5		A	Z*	A	Z*	A	Z
A6		A	Z*	A	Z*	A	Z
A8		A	Z*	A	Z*	A	Z
A9		A	Z*	A	Z*	A	Z
AB		A	Z*	A	Z*	A	Z
AC		A	Z*	A	Z*	A	Z
AF		A	Z*	A	Z*	A	Z
B0		Z	M	Z	Z	Z	
B1		Z	M	Z	Z	Z	
B2		Z	M	Z	Z	Z	
B3		Z	M	Z	Z	Z	
B4		Z	M	Z	Z	Z	
B5		Z	M	Z	Z	Z	
B6	Z						
B7		Z	A	Z	A*	Z	A
B8		Z	A			C	A
B9							
** BA		Z	A		A		A
B8			A*		A*	A	Z
BC			Z			A	Z
BD			A*		A*	A	A
BE							
BF							

* If Register Zero is selected to broadcast a constant, machine zero will be that constant.

** See Section 3.1.7 paragraph A.3

--- S U P E R C O M P U T E R O P E R A T I O N S ---

Table 3.1.7-1

Op Code	Instruction Designators G	X	A	Y	B	Z	C
C0		Z	A*	Z	A*	C	
C1		Z	A*	Z	A*	C	
C2		Z	A*	Z	A*	C	
C3		Z	A*	Z	A*	C	
C4		Z	A*	Z	A*	A	
C5		Z	A*	Z	A*	A	
C6		Z	A*	Z	A*	A	
C7		Z	A*	Z	A*	A	
C8			A		A	C	Z
C9			A		A	C	Z
CA			A		A	C	Z
CB			A		A	C	Z
CC		Z	A		A		A
CD							
CE	Z						
CF		Z	A	Z	A*	Z	Z
D0		Z	A*	Z	A*	C	A
D1		Z	A			C	A
D4		Z	A*	Z	A*	C	A
D5		Z	A			C	A
D6							
D7							
D8		Z	A			C	
D9		Z	A			C	
DA		Z	A			C	
DB		Z	A			C	
DC		Z	A*	Z	A*	C	
DD							
DE							
DF			M		M	C	A
E0							
E1							
E2							
E3							
E4							
E5							
E6							
E7							
E8							
E9							
EA							
EB							
EC							
ED							
EE							
EF							
F0		Z	A	Z	A	Z	A
F1		Z	A	Z	A	Z	A
F2		Z	A	Z	A	Z	A
F3		Z	A	Z	A	Z	A
F4		Z	A	Z	A	Z	A
F5		Z	A	Z	A	Z	A
F6		Z	A	Z	A	Z	A
F7		Z	A	Z	A	Z	A
F8		Z	A			Z	A
F9							
FA							
FB							
FC							
FD							
FE							
FF							

3.1.8 Real Time Counters

3.1.8.1 Free Running Clock

This clock consists of a free-running 47-bit counter and a positive sign bit for a total of 48 bits. It can be stored into register T using a "Transmit Real-Time Clock to T" (39) instruction. This counter increments at a one MHz rate.

3.1.8.2 Monitor Interval Timer

The "Monitor Interval Timer" is a 32-bit timer that decrements at a one MHz rate.

This timer can be loaded from Register R using the Transmit (R) to Monitor Interval Timer (0A) instruction, when the computer is in Monitor Mode. The timer can be activated by loading it with anything but all zeros. Once it is activated, it will decrement until it reaches zero or is deactivated. When the timer is decremented to zero, it will cause an external interrupt on channel 17 which must be processed like any other external interrupt.

The timer is deactivated by the following methods:

1. Master Clear
2. Loading with all zeros
3. Decremented to all zeros (when it is decremented to all zeros and caused an external interrupt, it will be inactive until loaded with some value other than zero).

3.1.8.3 Job Interval Timer

The Job Interval Timer is a 32-bit counter decrementing at a one MHz rate.

This clock can be loaded (in Job Mode only) from register R using a 3A (Transmit R to Job Interval Timer) instruction. Once loaded, the timer continues to decrement until either an exchange to monitor mode occurs, the timer decrements to zero, or the timer is loaded with a value of zero. If an exchange to monitor mode occurs, the decrementing of the Job

(continued)

3.1.8.3 (cont'd)

Interval Timer is stopped and the current contents of the timer are stored in the Invisible Package. When the execution of that job is resumed, the Job Interval Timer is loaded from the invisible package and resumes decrementing.

When the timer decrements to zero, bit 36 of the Data Flag Branch Register will be set. Thus, if the corresponding mask bit is set, a data flag branch would then occur during the next RNI.

The timer may be deactivated by loading it with a value of zero. This does not cause bit 36 of the Data Flag Branch Register to be set. Master clear will also deactivate the Job Interval Timer.

The Timer is deactivated by the following methods:

1. Master Clear
2. Loading with a value of zero
3. Decrementing to zero

The contents of the Job Interval Timer may be sampled by use of the 37 instruction (TRANSMIT JOB INTERVAL TIMER TO T). This does not deactivate the counter.

3.1.9 Virtual Addressing Mechanism

The virtual addressing mechanism provides a method of allotting central storage to jobs in the system.

3.1.9.1 Definitions Associated with Virtual Addressing

3.1.9.1.1 Monitor Mode and Job Mode

The CPU automatically goes into a hardware state called Monitor Mode at the time an interrupt is honored or at the time an Exit Force from a job is performed.

(continued)

3.1.9.1.1 (cont'd)

When in Monitor Mode, interrupts and the virtual addressing mechanism are disabled. This causes CPU addresses to be absolute addresses. Any interrupts which occur are saved until the Monitor program executes either an Idle or an Exit Force instruction.

The Idle instruction enables the interrupts and merely idles until an interrupt occurs.

The Exit Force instruction actually causes the hardware state to change to Job Mode and the job execution to start. During Job Mode, the interrupts are enabled and the virtual addressing mechanism is used by the CPU.

3.1.9.1.2 Page

A page consists of a block of contiguous words of central storage. All the words in a page are identified by a common page identifier. This identifier is an absolute address which locates a page in absolute storage. The following table identifies the page sizes.

Page Sizes (64-bit words)

Small	Large
512[D]	65,536[D]
2,048[D]	
8,192[D]	

The size of the small page is selected by bits 0 and 16 (bit 0 of Key 0 and 1) in the 3rd word (keys) of the invisible package. The bits are interpreted as follows:

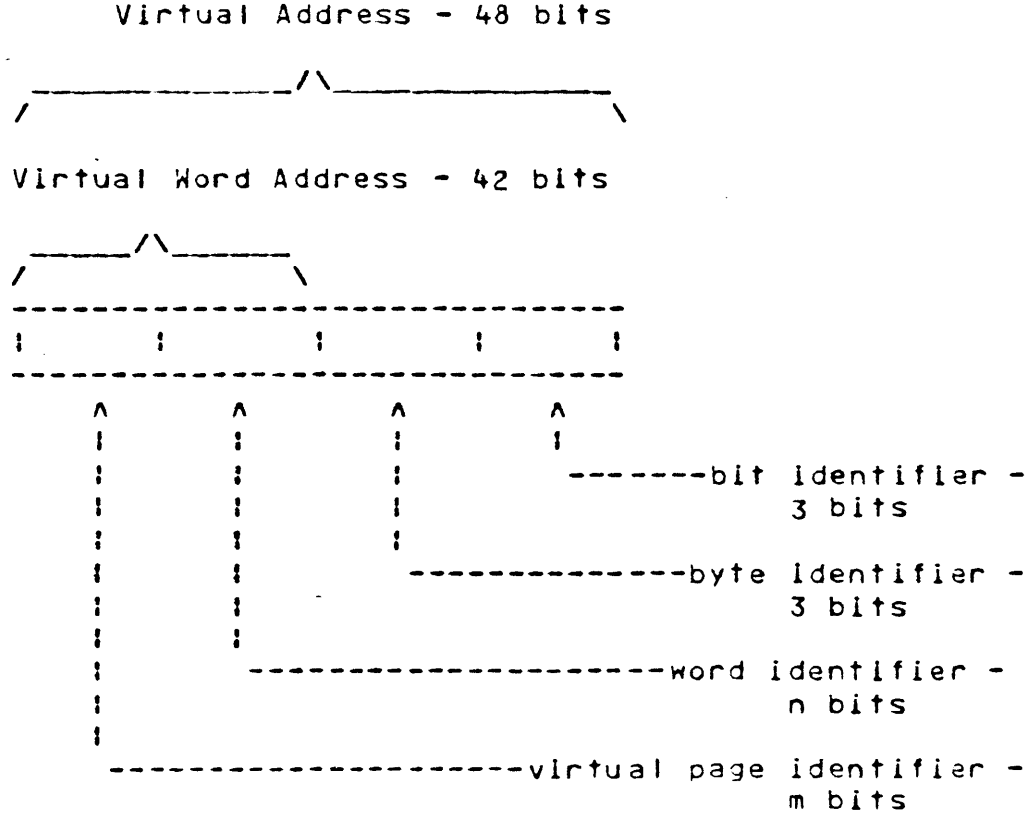
(continued)

3.1.9.1.2 (cont'd)

Bits		
1	16	
0	0	All small pages will be interpreted by the job as 512[D] words.
1	0	Small pages are 2,048[D] words.
1	1	Small pages are 8,142[D] words.
0	1	Undefined

Virtual Address

Addresses originating in the central processor (when not in Monitor Mode) are virtual addresses whose bits have the following interpretation:



(continued)

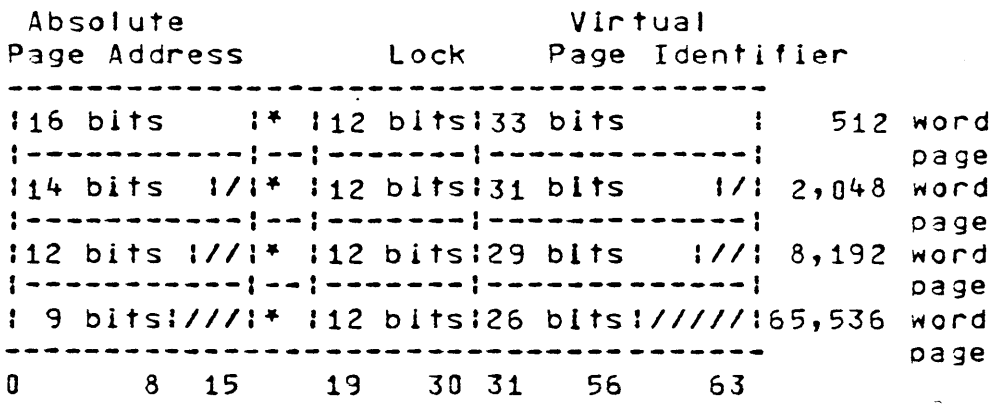
3.1.9.1.2 (cont'd)

where m and n are defined as shown in the following table:

Page Size (64-bit words)	m (bits)	n (bits)
512[D]	33	9
2,048[D]	31	11
8, 192[D]	29	13
65,536[D]	26	16

3.1.9.1.4 Associative Words

Associative words contain the information necessary to map a virtual address into an absolute address. They have one of the following three formats, depending on the page size:



Bits 14-15 and 62-63 are not used in associative word defining a 2,048 word pages.

Bits 12-15 and 60-63 are not used in associative word defining a 8,192 word pages.

Bits 9-15 and 57-63 are not used in associative word defining 65,536 word pages.

(continued)

3.1.9.1.4 (cont'd)

* Bits 16, 17, and 18.

3 bit code	Interpretation
000	end of the page table
001	null associative word
010	small page, has not been referenced by the CPU
011	large page, has not been referenced by the CPU
100	small page, has been referenced by the CPU
101	large page, has been referenced by the CPU
110	small page, has been alterec by the CPU.
111	large page, has been altered by the CPU

In the above table, the word referenced means that a job has made a storage reference to the page defined by the associative word.

The record of references and alterations contained in bits 16, 17 and 18 of associative word for page zero refer only to the upper half of the page.

Altered means that a CPU Job Mode program has done a write operation on at least one bit on the page defined by the associative word. When in Monitor Mode, the CPU does not use the associative hardware and alteration or referencing by the Monitor program is not recorded in the associative words.

3.1.9.1.5 Associative Registers (AR) and Space Table

Each computer has a fixed number of hardware associative registers which hold associative words for faster accessibility. The remainder of the associative words are in central storage starting at a fixed address. The portion of the total list of associative words which are in central storage is referred to as the space table. The contents of the hardware associative registers, when in storage, are stored beginning at absolute 4000[H].

(continued)

3.1.9.1.5 (cont'd)

Number of Associative Registers	Absolute Starting Address of Space Table
16	4400[H]

3.1.9.1.6 Page Table

The page table is the complete table of associative words and includes both the AR and the space table. These associative words define the pages currently allotted space in central storage.

3.1.9.1.7 Absolute Address

The absolute address is the combination of the absolute page address from the associative word in the page table and the word, byte and bit identifiers from virtual addresses. The following figure illustrates the construction of the abolute address for each page size.

Absolute Page Address	Word Identifier	Byte and Bit Identifiers	
16 bits	9 bits	6 bits	512 word page
14 bits	11 bits	6 bits	2,048 word page
12 bits	13 bits	6 bits	8,192 word page
9 bits	16 bits	6 bits	65,536 word page

The maximum storage address capacity of each computer limits the number of bits used for storage reference. Consequently, a number of left-most bits of the absolute page address are ignored and must be set to zero. The following table shows the number of left-most bits that are ignored for various storage size.

(continued)

3.1.9.1.7 (cont'd)

Maximum Storage Capacity (64-bit words)	Number of Left-Most Bits of Absolute Page Address Ignored
524,288	6
1,048,576	5
2,097,152	4
4,194,304	3
8,388,608	2

When in Monitor Mode, all CPU addresses are absolute addresses.

3.1.9.1.8 Lock

A lock is a 12-bit quantity contained in an associative word used to associate a page of central storage with a job or jobs which will reference the page.

3.1.9.1.9 Keys

Each job has four 12-bit keys assigned to it by the Monitor. If a job is to use a virtual page, the job must have the key matching the lock associated with that page.

* 0	Key 0	* 1	Key 1	* 2	Key 2	* 3	Key 3
4	15	20	31	36	47	52	63

*0, *1, *2, and *3 are four 4-bit usage lockout codes associated with Keys 0, 1, 2, and 3, respectively. These 4-bit codes have the following significance:

bit 0 - bit 0 of key 0 and 1 (bit 0 and 16) are used to define small page size (see Sec. 3.1.9.1.2) bit 0 of key 2 and 3 (bit 32 and 48) must be set to zero
bit 1 - if set, locks out CPU write operations
bit 2 - if set, locks out CPU read operations
bit 3 - if set, locks out CPU instruction references

(continued)

3.1.9.1.9 (cont'd)

If a key matches the lock of an associative word, but the usage attempted is locked out by that key's lockout bits, an access interrupt will occur.

See Section 3.1.10 for the location of a job's keys within the Invisible Package.

3.1.9.2 Operation of the Virtual Addressing Mechanism

A virtual address is sent from the central processor to the storage virtual addressing mechanism. The virtual addressing mechanism compares the virtual page identifier from the address against the virtual page identifier of each entry in the page table. If a match is found, and if the lock associated with the matching virtual page identifier is matched by one of the four keys possessed by the requesting job, a hit has been made. If a hit is made, the absolute page address associated with the hit-producing page table entry is attached to the word identifier from the central processor address. This combination forms the absolute address used to reference storage.

If the end of the page table is found and no hit has been made, a storage access interrupt occurs. If a hit is made, but the operation is prohibited by the usage lockout bits, a storage access interrupt occurs.

For a more detailed description of the page table search, see individual model specification as listed in Section 2.0.

3.1.9.3 Access Interrupts

A storage access interrupt occurs whenever a page referenced by a CPU job does not have its associative word in the page table or if an attempt is made by a CPU job to violate the usage code of a page as defined by the key matching the lock.

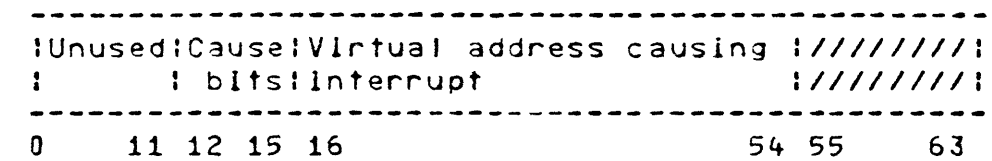
An access interrupt may be caused by any storage reference made by the CPU, even in the middle of a vector or string instruction. The virtual address causing the interrupt and information bits as to the

(continued)

3.1.9.3 (cont'd)

reason for the access interrupt are stored (along with the current contents of the invisible flags and invisible registers) into the Invisible Package.

An access interrupt will generate the following entry in word address E(H) of the job's Invisible Package:



Bits 0 through 11 are not used and will be set to zeros.

Definition of cause bits:

Bit 12	Bit 13	
0	1	Write operand violation attempted
1	0	Associative word not in the page table
1	1	*Associative word not in the page table and reference attempted was a write operation

Bit 14 = 1 Read operand violation attempted
Bit 15 = 1 Read instruction violation attempted

*Note that that is the only case where more than one of the cause bits would be set.

Bits 55 through 63 are undefined and may not always be zero.

Bits 16 through 54 contain the virtual sword address which caused the access interrupt.

The CPU then reverts to Monitor Mode and a branch is made to the absolute address contained in the right-most 48 bits of the Monitor's register 7.

The Monitor program takes care of allocating space for and/or procuring the requested page. After this is done, the Monitor can start the job exactly where it left off via an Exit Force instruction.

3.1.10 Exchange Operations and Invisible Package

The purpose of the exchange is to change the prime role of the CPU from Monitor Mode to Job Mode and from Job Mode back to Monitor Mode.

The exchange operation from Monitor to a Job is always accomplished with an Exit Force instruction. This causes the contents of the Invisible Package to be loaded into the appropriate registers; the mode to be changed from Monitor to Job enabling the virtual address mechanism and interrupts; and execution to be begun as specified by the Invisible Package. Note that this may be the restarting of a previously interrupted program.

The Exit Force instruction, the channel interrupt and the access interrupt are the three normal ways of getting from a job in Job Mode to the Monitor program in Monitor Mode. Attempting to execute a Monitor-type instruction in Job Mode or attempting to execute an undefined op-code comprises the fourth way to get into the Monitor mode. Except for the starting point in the Monitor program, the operation performed in getting to the Monitor are identical for the four. Sufficient information to restart this job is stored into the Invisible Package and the mode is changed from Job to Monitor. The Monitor program is executed starting at the absolute address contained in the right-most 48 bits of the Monitor's register 3, 5, 6, or 7.

Method of getting to the Monitor	Monitor register, the contents of which is used to set P
1. Attemot to perform an illegal instruction or a Monitor-type instruction in Job Mode*	Register 3
2. Attempt to perform an illegal instruction in Monitor Mode*	Register 4
3. Exit Force	Register 5
4. External Interrupt	Register 6
5. Storage Access Interrupt	Register 7

*See section 3.1.4.2.2

(continued)

3.1.10 (cont'd)

The right-most ten bits of the absolute starting address of the Invisible Package must be zeros.

The Monitor must set up an invisible package for each job. There is NO Invisible package for the Monitor program itself.

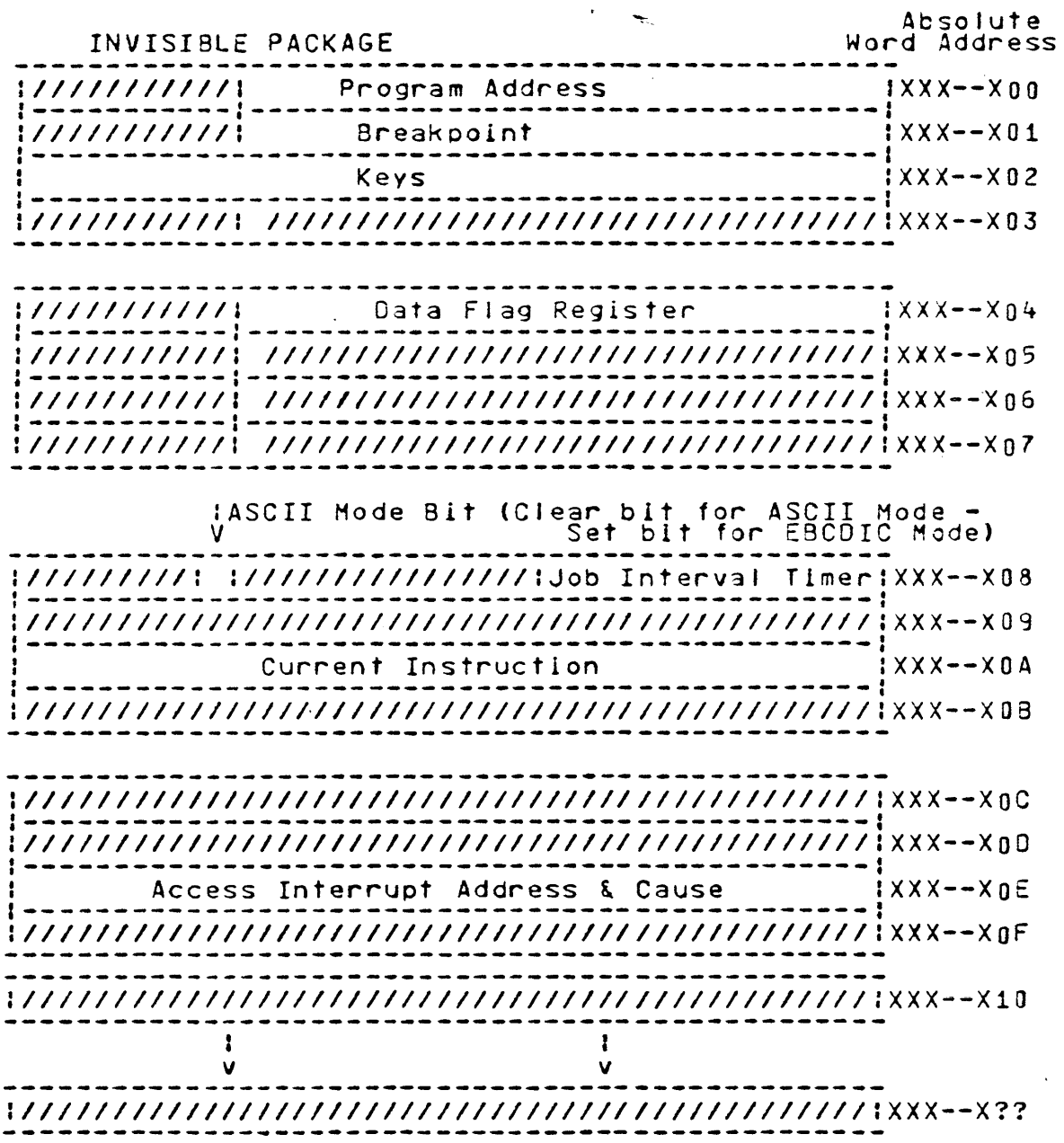
To start a job initially, the Monitor must clear the entire Invisible Package area except for the Keys and the Program Address areas.

If a job is to be re-entered, the Monitor should not alter any of the Invisible Package except for possibly the Keys.

For a more detailed description of the exchange operation, and the size of the invisible package (which may vary from model to model) see the applicable computer specification as listed in Section 2.0.

(continued)

3.1.10 (cont'd)



Each of the computers returns the same information in the non-crosshatched areas. The definition of the cross-hatched areas and size of the package are model dependent. For specific detail see the applicable machine specification as listed in Section 2.0.

3.2 Performance Characteristics

3.2.1 Instruction Descriptions

The instruction titles (3.2.1.1 - 3.2.1.256) are written in the following format:

3.2.1.XXX AA B CC DD NAME OF INSTRUCTION [AX]

where AA = the function code 00-FF[H]

B = the format types, 1-C

CC = the number of bits in the operand

1	single bit
8	bytes
32	half-words
64	words
E	either 32 or 64-bit
B	both 32 and 64-bit
NA	operand size not applicable

DD = the instruction type

	Blank Undefined
BR	Branch
IN	Index
LS	Logical String
MN	Monitor
NT	Non-Typical
RG	Register
ST	String
SV	Sparse Vector
VM	Vector Macro
VT	Vector

The G bit usage charts in the table of contents use the following symbols. Positions in the G bit usage charts without symbols are undefined and must be set to zero.

	G bit
E - Either 32 or 64-bit operands	0
C - Control vector	1
O - Offset	2
B - Broadcast	3, 4
S - Sign Control	5, 6, 7
X - Defined in Instruction	Any

(continued)

3.2.1 (cont'd)

Function Code	G-Bits 0	1	2	3	4	5	6	7
80	E	C	0	B	B	S	S	S
81	E	C	0	B	B	S	S	S
82	E	C	0	B	B	S	S	S
83		C	0	B	B			
84	E	C	0	B	B	S	S	S
85	E	C	0	B	B	S	S	S
86	E	C	0	B	B	S	S	S
87		C	0	B	B			
88	E	C	0	B	B	S	S	S
89	E	C	0	B	B	S	S	S
8A		C	0	B	B			
8B	E	C	0	B	B	S	S	S
8C	E	C	0	B	B	S	S	S
8D								
8E								
8F	E	C	0	B	B	S	S	S

Function Code	G-Bits 0	1	2	3	4	5	6	7
90	E	C	0	B				
91	E	C	0	B				
92	E	C	0	B				
93	E	C	0	B		S	S	
94	E	C	0	B	B			
95	E	C	0	B	B			
96		C	0	B				
97		C	0	B				
98	E	C	0	B				
99	E	C	0	B				
9A	E	C	0	B				
9B	E	C	0	B	B			
9C		C	0	B				
9D	E	C	0	B	B	X	X	X
9E								
9F								

(continued)

3.2.1 (cont'd)

Function Code	G-Bits 0	1	2	3	4	5	6	7
A0	E	X	X	B	B	S	S	S
A1	E	X	X	B	B	S	S	S
A2	E	X	X	B	B	S	S	S
A3								
A4	E	X	X	B	B	S	S	S
A5	E	X	X	B	B	S	S	S
A6	E	X	X	B	B	S	S	S
A7								
A8	E	X	X	B	B	S	S	S
A9	E	X	X	B	B	S	S	S
AA								
AB	E	X	X	B	B	S	S	S
AC	E	X	X	B	B	S	S	S
AD								
AE								
AF	E	X	X	B	B	S	S	S

Function Code	G-Bits 0	1	2	3	4	5	6	7
B0	X	X	X	X	X	X	X	X
B1	X	X	X	X	X	X	X	X
B2	X	X	X		X	X	X	X
B3	X	X	X		X	X	X	X
B4	X	X	X		X	X	X	X
B5	X	X	X		X	X	X	X
B6								
B7	E				B		X	X
B8	E	C	0					
B9	E		X	X	X			
BA	E						X	X
BB	E			B	B			
BC	E	X						
BD	E			B	B			X
BE								
BF								

(continued)

3.2.1 (cont'd)

Function Code	G-Bits 0	1	2	3	4	5	6	7
C0	E	C		B	B			
C1	E	C		B	B			
C2	E	C		B	B			
C3	E	C		B	B			
C4	E			B	B			
C5	E			B	B			
C6	E			B	B			
C7	E			B	B			
C8	E	C	X					
C9	E	C	X					
CA	E	C	X					
CB	E	C	X					
CC	64							X
CD								
CE								
CF	E				B	S	S	S

Function Code	G-Bits 0	1	2	3	4	5	6	7
D0	E	C	0	B	B			
D1	E	C	0					
D2								
D3								
D4	E	C	0	B	B			
D5	E	C	0					
D6								
D7								
D8	E	C				S		
D9	E	C				S		
DA	E	C						
DB	E	C						
DC	E	C						
DD								
DE								
DF	E	C	0					

3.2.1.1 00 4 NA MN IDLE

When in Monitor Mode, enable the external interrupts and idle until an external interrupt occurs. The R, S and T designators are undefined and must be set to zero.

3.2.1.2 01 ILLEGAL

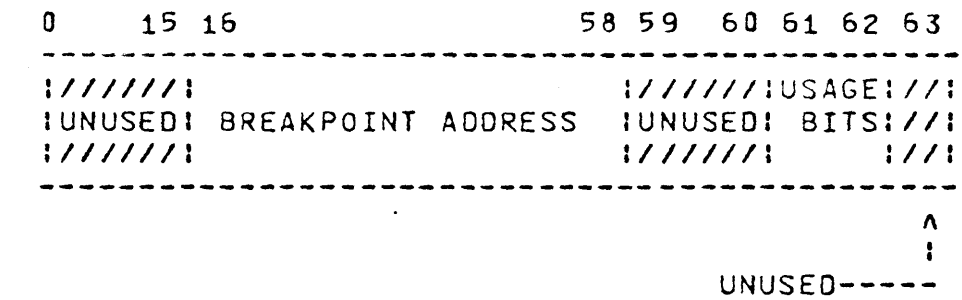
3.2.1.3 02 ILLEGAL

3.2.1.4 03 KEYPOINT - MAINTENANCE

This instruction is for use only with the Instrumentation Monitoring System. This operation is model dependent, see applicable machine specification listed under section 2.0.

3.2.1.5 04 4 64 NT BREAKPOINT-MAINTENANCE

The breakpoint instruction transfers R to the breakpoint register. The breakpoint register is a maintenance and program debugging aid and is saved in the job's Invisible Package.



The breakpoint function compares the addresses of specified categories of requests with the breakpoint address. If a match occurs, bit 47 of the Data Flag Branch register is set.

Usage bits 61 and/or 62 may be set to specify the breakpoint function for CPU write operands and/or CPU read operands respectively.

USAGE BITS

61 - BREAKPOINT ON CPU WRITE OPERANDS
62 - BREAKPOINT ON CPU READ OPERANDS

(continued)

Page Missing From Original

3.2.1.7 (cont'd)

This instruction is only enabled during Monitor Mode. In Job Mode it becomes a no op.

The modes are set up by executing this instruction with a "1" in the appropriate R designator bit and are cleared by executing the instruction with a "0" in the same bit location.

The R designator bits are defined below:

R DESIGNATOR BIT

8	Disable error correction on all Read buses.
9-15	Checkword bits to be complemented.

Programmer Note: These bits must be set to zero before any Monitor to Job exchange Operation. If these bits are not set to zero via an 06 instruction, the connection network could produce invalid data on the Read and invalid data could be written into memory.

The S and T designators are undefined.

A description of each of these faults can be found in applicable model specification under section 2.0.

Test Operation

SECDED FAULTS

The test is initiated by executing an 06 instruction with bits 9 through 15 selected of the R designator to complement the respective checkword bits of half-words 0, 1, 2, and 3 on the Write Scalar bus to central memory. By appropriate selection of data bits and complementation of checkword bits when writing in memory, one should be able to generate SECDED faults on all Read buses. This should allow

(continued)

3.2.1.7 (cont'd)

complete checking of the Read SECDED hardware and also the fault recording hardware for type and address of the fault.

The forced complementing of the checkword bits is discontinued by executing an 06 instruction with bits 9 through 15 of the R designator.

3.2.1.8 07 ILLEGAL

3.2.1.9 08 4 NA MN INPUT/OUTPUT PER R

When in Monitor Mode: Activate the channel flag designated by the R designator and exit to the next sequential instruction. If the R designator specifies a non-existent channel, the operation of this instruction is undefined.

The S and T designators are undefined and must be set to zero.

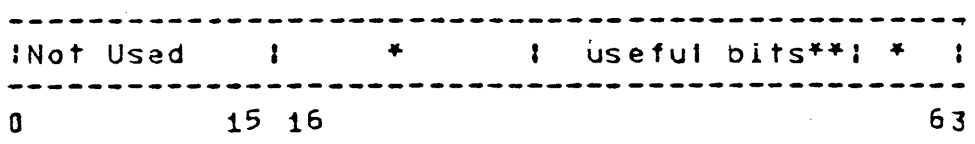
3.2.1.10 09 4 64 BR EXIT FORCE

From a job to the Monitor: Exchange to the Monitor program. A hardware branch is then taken to the address defined by the right-most 48 bits of the Monitor's register 5. For this case, the R, S and T designators are undefined and must be set to zero.

From the Monitor to a job: Exchange to the job whose Invisible Package is located starting at the absolute bit address contained in register T and whose virtual page zero (equivalent to starting address of register file to be loaded) starts at the absolute bit address contained in register S. For this case, the R designator is undefined and must be set to zero. If either the S designator or the contents of register S are equal to zero, the job's register file and the monitor's register file are identical.

(continued)

3.2.1.10 (cont'd)



Register S - I Bits 0-15 are not used

* These bits must always be set to zero or this instruction is undefined.

**The amount of central storage actually on a system will determine the number of useful bits on a given system.

The number of useful and unused bits is model dependent and related to memory size. See applicable model specification listed in section 2.0.

3.2.1.11 0A 4 64 MN TRANSMIT (R) TO MONITOR INTERVAL TIMER

When in Monitor Mode, transmit bits 32 through 63 of 64-bit register R to the Monitor Interval Timer (see Section 3.1.8). The left-most 32 bits of register R are ignored. The S and T designators are undefined and must be set to zero.

3.2.1.12 0B ILLEGAL

3.2.1.13 0C 4 64 MN STORE ASSOCIATIVE REGISTERS

3.2.1.14 0D 4 64 MN LOAD ASSOCIATIVE REGISTERS

When in Monitor Mode, Store/Load the contents of the associative registers into/from absolute addresses 4000(H), etc. The R, S and T designators are undefined and must be set to zero. The contents of the associative registers are undefined following the execution of the STORE ASSOCIATIVE REGISTERS instruction.

3.2.1.15 0E 4 64 MN TRANSLATE EXTERNAL INTERRUPT

Each bit in the External Interrupt Register (EIR) is associated with an external I/O channel, or the Monitor Interval Timer.

External Interrupt Register Bit	Assignment
0	Not Available
1	I/O Channel 1
2	! 2
3	! 3
4	! 4
5	! 5
6	! 6
7	! 7
8	! 8
9	! 9
10	! 10
11	! 11
12	! 12
13	! 13
14	! 14
15	v 15
16	I/O Channel 16
17	Monitor Interval Timer 17

Translate the lowest numbered bit set in the EIR into its associated five-bit code and transmit this code to the right-most five bits of register T. The left-most 59 bits of register T are cleared to zero.

Examine the EIR and if only one bit is set, the branch condition is met. The branch, if taken, is to (S) + (R) where (S) is an index in half-words and (R) is the base address.

The exit, be it a branch or not, clears the bit (and only that bit) in the EIR corresponding to the channel designator which was transmitted to register T.

If the T and S designators are equal, the interrupting channel designator will also be the branch index.

Bit zero of the EIR will <u>never</u> be set as it is reserved for maintenance purposes.

If no bit in the EIR is set, this instruction sets T to all zeroes and no branch is taken.

3.2.1.16 OF 4 64 MN LOAD KEYS FROM(R),TRANSLATE ADDRESS(S) TO (T)

When in Monitor Mode load the four keys found in register R into the virtual address key registers. The virtual address found in register S is then translated into an absolute bit address using the four keys just loaded and the associative words of the page table. This absolute bit address is stored in the right-most 48 bits of register T. If no translation is possible before the end of the page table is encountered, the right-most 48 bits of T are set to zero. The associative word actually used to make the translation is left in the top associative register (associative register zero). The page table is dynamically pushed down, if necessary, when searching for the associative word used to make the translation. This instruction uses the page table as contained in the Associative Registers and the Space table in memory. The locations in memory corresponding to the Associative Reg isters (see the OC and OD instructions) will not be re ferenced during the execution of the OF instruction. The left-most 16 bits of register S are transmitted to the corresponding position in register T.

The 3-bit size, alteration and reference code in the associative word is not changed by this instruction.

The entire address range (including bit addresses 0 through 3FFF base 16) are acceptable inputs to the OF instruction.

////	Key 0	////	Key 1	////	Key 2	////	Key 3	Register R
bit 0	4	15	20	31	36	47	52	63

Bits 1-3, 17-19, 32-35, and 48-51 of register R are not used by this instruction.

	absolute bit address	Register T
bit 0	15 16	63

	virtual address	Register S
bit 0	15 16	63

(continued)

3.2.1.16 (cont'd)

Bits 0 and 16 of the key word must be appropriately set/clear to indicate the desired small page size.

3.2.1.17 10 A 64 RG CONVERT BDC TO BINARY, FIXED LENGTH

Convert the packed BCD number in register R to a signed (two's complement) binary number and place the result into the right-most 48 bits of register T. The conversion is undefined for binary results greater than +(2**47)-1 or less than -((2**47); thus the largest decimal number that may be converted is ±140,737,488,355,327. The ASCII/EBDIC sign code for the BCD number is in bits 60-63 of register R.

Data flag bit 39 will be set for numbers outside this range.

If the input number is not a valid BCD number, the results are undefined. Bits 0-15 of Register T will be cleared to zero.

3.2.1.18 11 A 64 RG CONVERT BINARY TO BCD, FIXED LENGTH

Convert the right-most 48 bits (two's complement binary number) of register R to a packed BCD number and place the result in register T. The result is a number having 15 digits (4 bits per digit plus the sign in the lower bits - bits 60-63). The binary range is + ((2**47)-1) too -(2**47). During job mode, the sign bits generated are conditioned by the ASCII/EBCDIC bit in the job's invisible package. During monitor mode, only ASCII codes will be generated.

3.2.1.19 12 7 64 NT LOAD BYTE; (T) PER (S), (R)

3.2.1.20 13 7 64 NT STORE BYTE; (T) PER (S), (R)

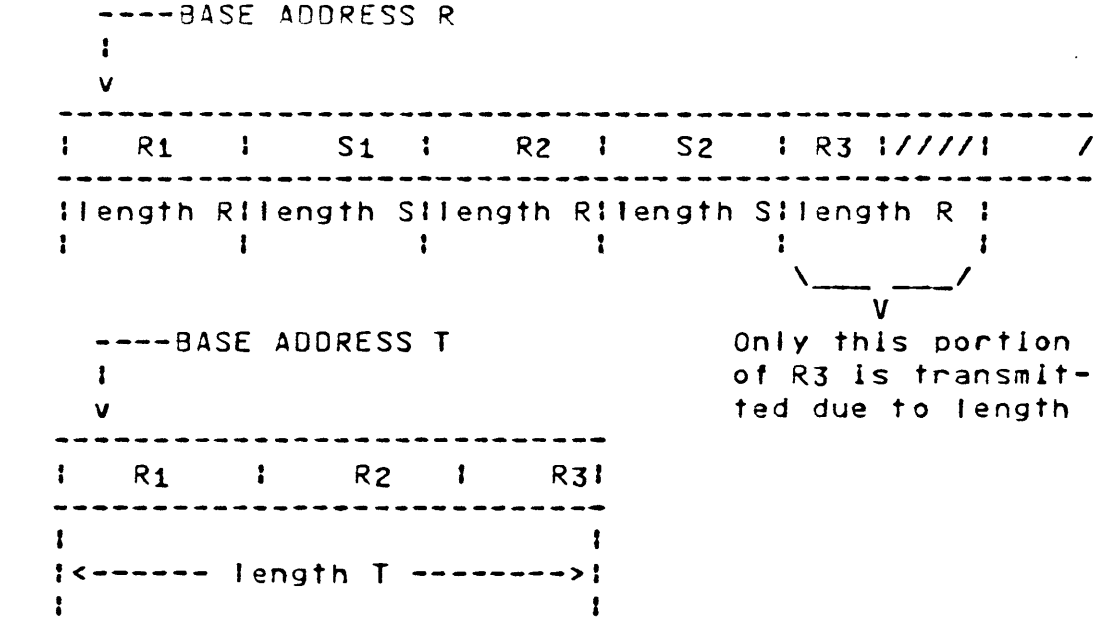
Load/store a byte from/into the address specified by (R) + (S), where (R) is the base address and (S) is an item count of bytes into/from bits 56 through 63 of register T. The remaining bits of T are cleared on a load and ignored on a store.

3.2.1.21 14 7 1 NT BIT COMPRESS

Compress bit field R per length S into bit field T.

The length in bits of segments in the source field to be transmitted to the destination field and the base address of the source field are found in the left-most 16 bits and the right-most 48 bits of register R, respectively. The length in bits of segments in the source field to be skipped is found in the left-most 16 bits of register S. The length and base address of the destination field are found in the left-most 16 bits and the right-most 48 bits of register T, respectively.

Transmit from left to right a portion of the R field equal to length R to the T field and skip the number of R field bits equal to length S. This operation is repeated until the T field is exhausted. If the field length specified by R or T is zero, the instruction is treated as a no op.



3.2.1.22 15 7 1 NT BIT MERGE

Merge bit fields R and S into bit field T.

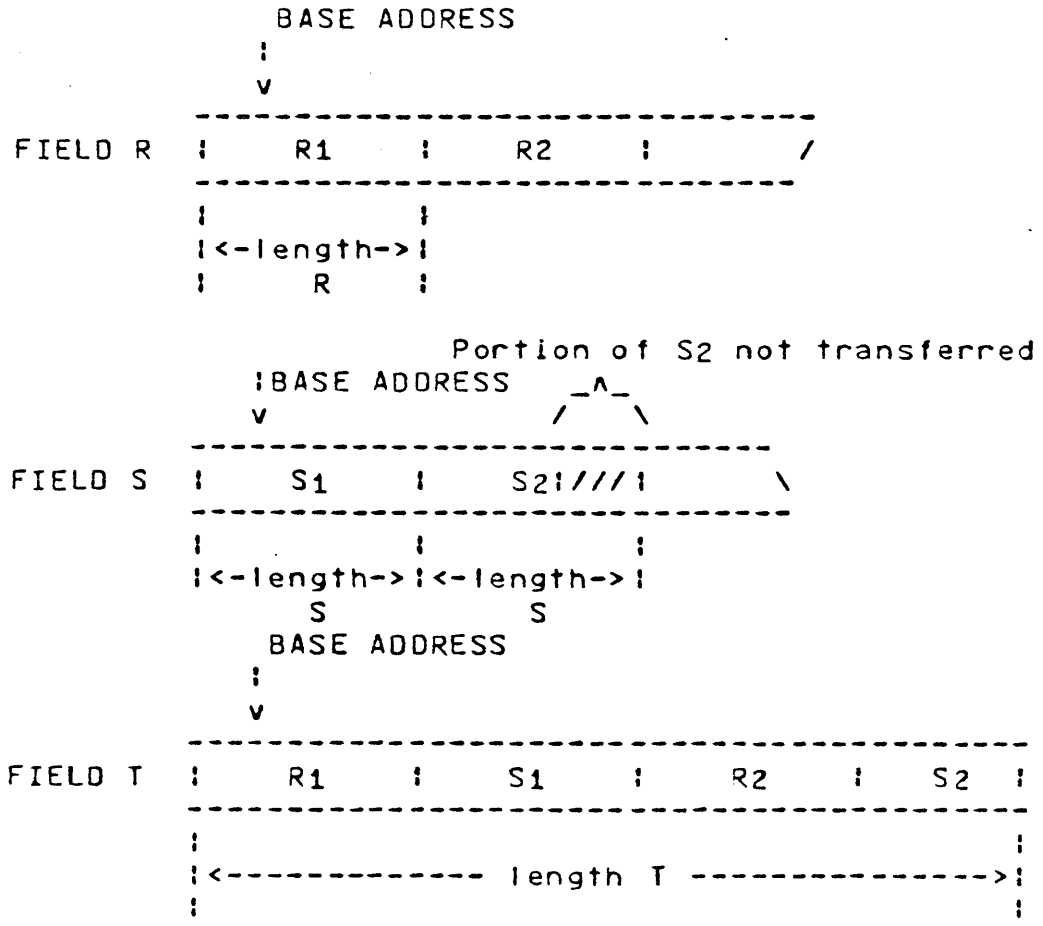
The length in bits of segments of the R field to be merged and the base address of the R field are found

(continued)

3.2.1.22 (cont'd)
in the left-most 16 bits and the right-most 48 bits of register R, respectively. The length in bits of segments of the S field to be merged and the base address of the S field are found in the left-most 16 bits and the right-most 48 bits of the S register, respectively. The length in bits and the base address of the destination field are found in the left-most 16 bits and the right-most 48 bits of register T, respectively.

Transmit from left to right a segment of the R field equal to length R followed by a segment of the S field equal to length S to field T. This operation is repeated until the T field is exhausted.

If bits 16 thru 63 of S are zero, logical zeros will be placed in their respective fields in the T field. If the field length specified by R, S or T is zero, the instruction is treated as a no-op.



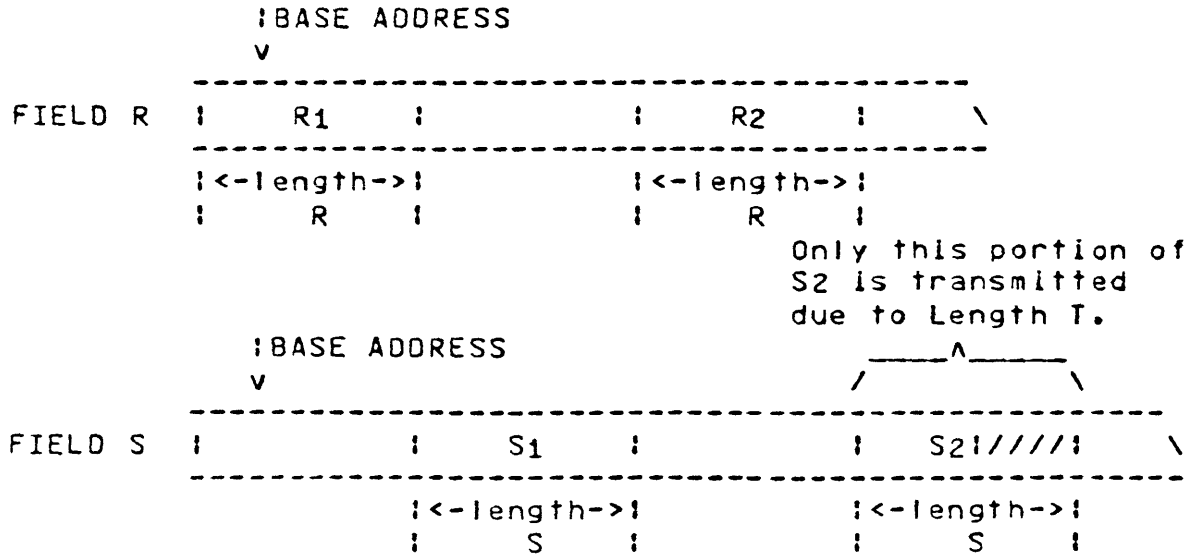
3.2.1.23 16 7 1 NT BIT MASK

Mask bit fields R and S into bit field T.

The length in bits of segments of the R field to be masked and the base address of the R field are found in the left-most 16 bits and the right-most 48 bits of register R, respectively. The length in bits of segments of the S field to be masked and the base address of the S field are found in the left-most 16 bits and the right-most 48 bits of the S register, respectively. The length in bits and the base address of the destination field are found in the left-most 16 bits and the right-most 48 bits of register T, respectively.

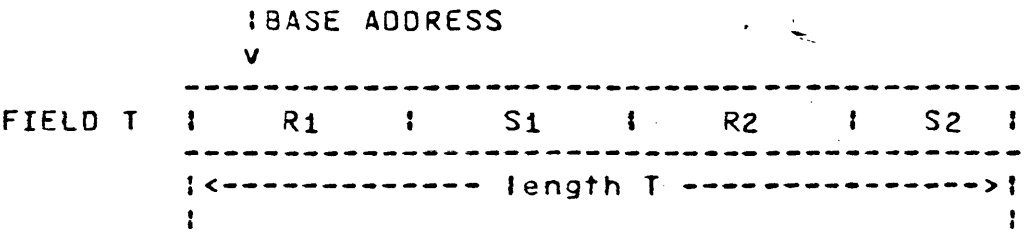
Transmit from left to right a segment equal to length R starting at the base address of field R to field T. Next transmit to field T a segment of Field S equal to length S starting at the base address of S plus length R. The next segment of R is transmitted to field T from address R plus length R plus length S. This operation is repeated until the T field is exhausted.

If bits 16 thru 63 of S are zero, logical zeros will be placed in their respective fields in the T field. If the field length specified by R, S, or T is zero, this instruction is treated as a no-op.



(continued)

3.2.1.23 (Cont'd)

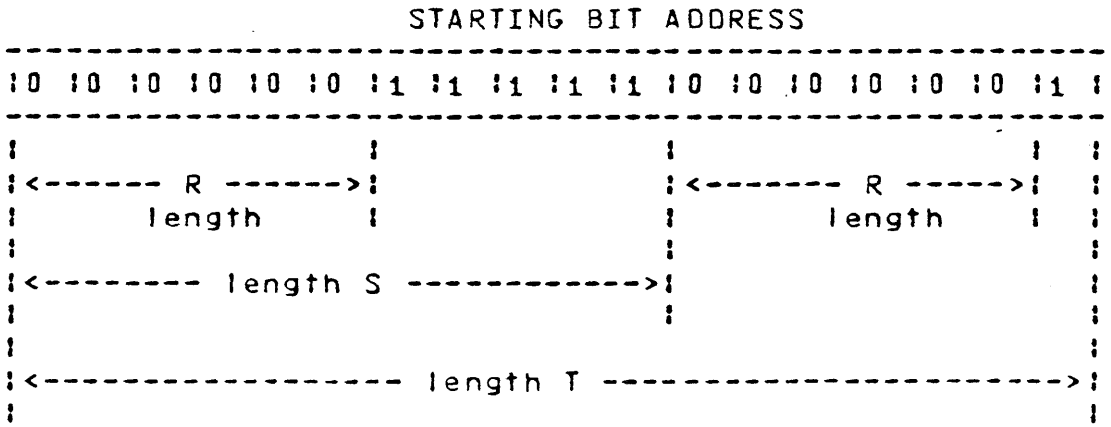


3.2.1.24 17 ILLEGAL
3.2.1.25 18 ILLEGAL
3.2.1.26 19 ILLEGAL
3.2.1.27 1A ILLEGAL
3.2.1.28 1B ILLEGAL

3.2.1.29 1C 7 1 NT FORM REPEATED BIT MASK WITH LEADING ZEROS

Form a repeated mask in field T. The mask is a string of zeros followed by a string of ones.

The length (in bits) of the string of zeros is found in the left-most 16 bits of register R. The length (in bits) of the repeated mask is found in the left-most 16 bits of register S. The length (in bits) and the starting address of field T are found in the left-most 16 bits and the right-most 48 bits of register T, respectively. The instruction terminates when the T field is exhausted. If length R is greater than length S, the instruction is undefined. If length R is equal to length S, a string of zeros is formed.

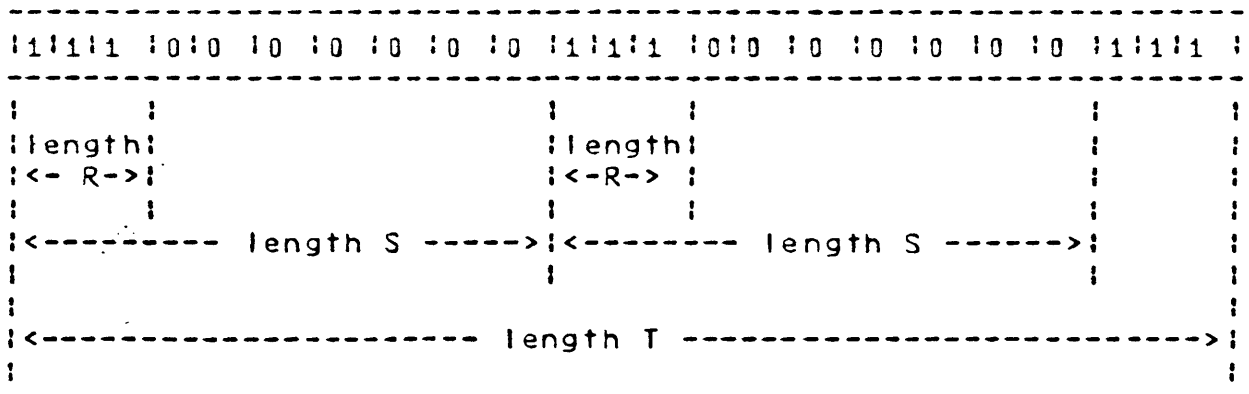


For length R equal to zero, a string of ones is formed. For length S equal to zero, the instruction is performed as a no-op.

3.2.1.30 1D 7 1 NT FORM REPEATED BIT MASK WITH LEADING ONES

Form a repeated mask in field T. The mask is a string of ones followed by a string of zeros.

The length (in bits) of the string of ones is found in the left-most 16 bits of register R. The length (in bits) of the repeated mask is found in the left-most 16 bits of register S. The length (in bits) and the starting address of field T are found in the left-most 16 bits and the right-most 48 bits of register T, respectively. The instruction terminates when the T field is exhausted. If the field length specified by S is zero, the instruction performs as a no-op. If length R is greater than length S, the instruction is undefined. If length R is equal to length S, a string of ones is formed. For length R equal to zero, a string of zeros is formed.



3.2.1.31 1E 7 1 NT COUNT LEADING EQUALS

The bits of field R, starting with the bit to the immediate right of the left-most bit of the field, are scanned from left to right until a bit unequal to the left-most bit is encountered. The count of the number of bits equal to the left most bit is stored in the right-most bits of register T.

The length (in bits) and the base address of the field are contained in the left-most 16 bits and the right-most 48 bits of register R, respectively. Register S contains an index in bits which is added to the base address to form the starting bit address

(continued)

3.2.1.31 (Cont'd)

of the field. The instruction terminates when either a bit unequal to the left-most field bit is encountered, or when the entire field has been scanned. In the latter case, the count stored will be the field length minus one.

```
                        |-- STARTING ADDRESS
                        |    (left-most bit)
                        v
      ---------------------------------------------------
      |0 0 0 1 1 1 1 1 |1|1 1 1 1 1 1 1 1 1 1 1|0 1 1 0 1
      ---------------------------------------------------
                         |<--the count stored->|
                         -----------------------
(T) = |0 0 0 0|0 ----------------> 0 0 B|
      -----------------------------------
      0        16                      63
```

The entire T register is cleared before the count is stored into it.

Data Flag bit 53 is cleared during initiation of this instruction and then set to a one if the left-most bit was a one.

3.2.1.32 1F 7 1 NT COUNT ONES IN FIELD R, COUNT TO (T)

The bits of field R are scanned from left to right and the number of binary one bits counted. This count is stored in the right-most bits of register T.

The length (in bits) and the base address of the field are contained in the left-most 16 bits and the right-most 48 bits of register R, respectively. Register S contains an index in bits which is added to the base address to form the starting bit address of the field. The instruction terminates when the entire field has been scanned.

The entire T register is cleared before the count is stored into it.

3.2.1.33 20 8 32 BR BRANCH IF (R) EQ (S) (32 BIT FP.)
3.2.1.34 21 8 32 BR BRANCH IF (R) NE (S) (32 BIT FP.)
3.2.1.35 22 8 32 BR BRANCH IF (R) GE (S) (32 BIT FP.)

3.2.1.36 23 8 32 BR BRANCH IF (R) LT (S) (32 BIT FP.)

Conditionally branch to the address in 64-bit register T.

R and S are 32-bit registers containing floating point operands.

The S operand is subtracted from the R operand. The branch decision is made on the result of this subtract according to the "floating point compare rules" in 3.1.4.5.

Data flag: bit 46.

3.2.1.37 24 8 64 BR BRANCH IF (R) EQ (S) (64 BIT FP.)
3.2.1.38 25 8 64 BR BRANCH IF (R) NE (S) (64 BIT FP.)
3.2.1.39 26 8 64 BR BRANCH IF (R) GE (S) (64 BIT FP.)
3.2.1.40 27 8 64 BR BRANCH IF (R) LT (S) (64 BIT FP.)

Conditionally branch to the address in 64-bit register T.

R and S are 64-bit registers containing floating point operands.

The S operand is subtracted from the R operand. The branch decision is made on the result of this subtract according to the "floating point compare rules" in 3.1.4.5.

Data flag bit 46.

3.2.1.41 28 7 8 NT SCAN EQUAL

Scan field T indexed by S, from left to right until the first byte equal to byte R is found. Stop the scan and increment index S by the number of unequal bytes scanned. If no byte is found equal to byte R, the S index is incremented by the number of bytes in the T field.

The length in bytes of the operation and base address of the operation are found in the left-most 16 bits and the right-most 48 bits of register T, respectively. The right-most 48 bits of register S contains a signed index. Byte R is found in the instruction word. Index S is an item count in bytes

(continued)

3.2.1.41 (Cont'd)

and is left-shifted three places before it is added to the base address.

Data flag bit 53 is set if no equal is found.

3.2.1.42 29 ILLEGAL

3.2.1.43 2A 6 64 RG ENTER LENGTH OF (R) WITH I (16 BITS)

Transfer the right-most 16 bits of this instruction to the left-most 16 bits of register R. Leave the right-most 48 bits of register R unchanged.

3.2.1.44 2B 4 64 RG ADD TO LENGTH FIELD

Add bits 00 through 15 of register R to bits 48 through 63 of S and store the result in bits 00 through 15 of register T. Bits 16 through 63 of register R are transferred to bits 16 through 63 of register T.

3.2.1.45 2C 4 64 RG LOGICAL EXCLUSIVE OR (R),(S), TO (T)
3.2.1.46 2D 4 64 RG LOGICAL AND (R),(S), TO (T)
3.2.1.47 2E 4 64 RG LOGICAL INCLUSIVE OR (R),(S), TO (T)

These instructions perform the indicated logical functions listed below. The function occurs bit by bit on the 64-bit operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

R	S	EXCLUSIVE OR R-S	AND R.S	INCLUSIVE OR R+S
0	0	0	0	0
0	0	1	0	1
1	0	1	0	1
1	1	0	1	1

If the R or S designators equal zero, register zero will contain machine zero.

3.2.1.48 2F 9 1 BR REGISTER BIT BRANCH AND ALTER

This instruction examines bit 63 of register T. As specified by the G designator a branch is made to the address contained in the right-most 48 bits of register S. The branch is made according to G bits 0 and 1 as follows:

G0	G1	
0	0	do not branch
0	1	branch unconditionally
1	0	branch if the object bit was a one
1	1	branch if the object bit was a zero

After the branch decision has been made and regardless of what that decision was, the object bit is altered according to G bits 2 and 3 as follows:

G2	G3	
0	0	do not alter the object bit
0	1	toggle the object bit to the other state
1	0	set the object bit to a one
1	1	clear the object bit to a zero

If the branch is to be taken, the branch address will be determined as follows:

G bit 5 = 0 Register S contains the branch address.

G bit 5 = 1 Branch to the address formed by adding (G bit 6 = 0) or subtracting (G bit 6 = 1) the S designator (used as a half-word item count) shifted left 5 places to the program address register.

3.2.1.49 30 7 64 RG SHIFT (R) PER S TO (T)

This instruction shifts the 64-bit operand from the register designated by R and stores the result into the register designated by T. The S designator specifies the type and amount of the shift. If the S designator is in the range from 0 through 3F base 16 (0 through 63 base 10), the operand from register R is shifted left end-around the number of specified

(continued)

3.2.1.49 (Cont'd)

places and then stored in register T. If the S designator is in the range from FF through C1 base 16 (-1 through -63 base 10), the operand from register T is shifted right with sign extension and then stored into register T. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the S designator. If for example, S is equal to FE base 16, the operand from register R shifts right two places. If the S designator is greater than 3F base 16 or less than C1 base 16, the results of this instruction are undefined.

If the R designator is equal to zero, register zero will provide machine zero. This instruction does not test for machine zero or indefinite or set any data flags.

3.2.1.50 31 7 64 BR INCREASE(R) AND BRANCH IF(R) ≠ 0

Increment the contents of the right-most 48 bits of register R by one. The upper 16 bits of register R are not altered and arithmetic overflow is ignored.

If the result from above is 48 zeros, go to the next sequential instruction. If the 48-bit result from above is non-zero, branch to (S) + (T) where (S) is an item count of half-words and (T) is the base address. The resulting address for the branch is undefined if the R designator is equal to either the S designator or the T designator.

3.2.1.51 32 9 1 BR BIT BRANCH AND ALTER

Register S contains the address of the object bit. This instruction reads up the word containing the object bit and examines the bit. The branch is then made according to G bits 0 and 1:

G0	G1	
0	0	do not branch
0	1	branch unconditionally
1	0	branch if the object bit was a one
1	1	branch if the object bit was a zero

(continued)

3.2.1.51 (Cont'd)

After the branch decision has been made and regardless of what that decision was, the object bit is altered according to G bits 2 and 3 as follows:

G2	G3	
0	0	do not alter the object bit
0	1	toggle the object bit to the other state
1	0	set the object bit to a one
1	1	clear the object bit to a zero

NOTE: If G0 and G2 and G3 = 0, do not reference the object bit at all.

If (G0 = 1) and (G2 and G3 = 0) read, but do not write the object bit.

If the branch is to be taken, the branch address will be determined as follows:

G bit 5 = 1 Branch to the address formed by adding (G bit 6 = 0) or subtracting (G bit 6 = 1) the T designator (used as a half-word item count) shifted left 5 places to the program address register.

3.2.1.52 33 B 1 BR DATA FLAG REGISTER BIT BRANCH AND ALTER

I is a six-bit designator specifying an object bit in the data flag register.

The object bit in the data flag register is examined and the decision to branch is made according to G bits 0 and 1.

G0	G1	
0	0	do not branch
0	1	branch unconditionally
1	0	branch if the object bit was a one
1	1	branch if the object bit was a zero

(continued)

3.2.1.52 (Cont'd)

After the branch decision has been made and regardless of what that decision was, the object bit is altered according to G bits 2 and 3 as follows:

G2	G3	
0	0	do not alter the object bit
0	1	toggle the object bit to the other state
1	0	set the object bit to a one
1	1	clear the object bit to a zero

Programmer Note: It is meaningless to try to alter bits in the product field (bits 0-15) since the product field is strictly a function of the appropriate data flag and flag mask bits.

If the branch is to be taken, the branch address will be determined as follows:

G bit 5 = 0	Register T contains the branch address
G bit 5 = 1	Branch to the address formed by adding (G bit 6 = 0) or subtracting (G bit 6 = 1) the T designator (used as a half-word item count) shifted left 5 places to the program address register.

Since the 33 instruction may begin execution without waiting until the machine has completed all operations (for example, the scalar divide's data flags may not have reached the Data Flag Register), the data flag bits may set on any minor cycle during or after execution of the 33 instruction. Consequently, any data flag bits that set after the object bit is sampled will not affect the operation of the 33 instruction, but will be retained in the Data Flag Register for follow on sampling.

3.2.1.53 34 4 64 RG SHIFT(R) PER (S) TO (T)

This instruction shifts the 64-bit operand from the register designated by R and stores the result into the register designated by T. The register designated by S specifies the type and amount of the

(continued)

3.2.1.53 (Cont'd)

shift. If the right-most byte of register S is in the the range from 0 through 3F base 16 (0 through 63 base 10), the operand from register R is shifted left end-around the number of specified places and then stored into register T. If the right-most byte of register S is in the range from FF through C1 base 16 (-1 through -63 base 10), The operand from register R is shifted right with sign extension and then stored into register T. For this case, bit zero of the operand from register R is considered to be the sign bit of the shifted operand. The number of right shifts is equal to the two's complement of the right-most byte of register S. If the right-most byte of register S is greater than 3F or less than C1 base 16, the results of this instruction are undefined. The left-most seven bytes of register S are ignored.

If the R designator is equal to zero, register zero will provide machine zero. This instruction does not cause a test for machine zero or indefinite or set any data flags.

3.2.1.54 35 7 64 BR DECREASE (R) AND BRANCH IF (R) ≠ 0

Decrement the contents of the right-most 48 bits of register R by one. The upper 16 bits of register R are not altered and arithmetic overflow is ignored.

If the result from above is 48 zeros, go to the next sequential instruction. If the 48-bit result from above is non-zero, branch to (S) + (T) where (S) is an item count of half-words and (T) is the base address. The resulting address for the branch is undefined if the R designator is equal to either the S designator or the T designator.

3.2.1.55 36 7 64 BR BRANCH AND SET(R)TO NEXT INSTRUCTION

After storing the address of the next sequential instruction into register R, branch to (S) + (T) where (S) is an item count of half words and (T) is the base address. Bits 0 through 15 of register R are forced to zeros. Bits 59 through 63 of register R are undefined. If the R designator is equal to the S designator the results of this instruction are undefined.

(continued)

3.2.1.55 (Cont'd)

NOTE: If S=0, and R=T, this instruction sets register R to the half-word address of the next instruction and the program continues at the next instruction. This is a way to sample the program address register (P).

3.2.1.56 37 A 64 NT TRANSMIT JOB INTEVAL TIMER TO (T)

Transmit the contents of the Job Interval Timer into bits 32-63 of register T. Bits 0-31 are cleared to zero. The R and S designators are undefined and must be set to zero. This instruction does not deactivate the timer.

When executed in Monitor Mode, the operation of this instruction is undefined.

3.2.1.57 38 A 64 IN TRANSMIT (R BITS (00-15) TO T BITS (00-15)

Replace the left-most 16 bits of register T with the left-most 16 bits of register R.

3.2.1.58 39 A 64 NT TRANSMIT REAL-TIME CLOCK TO (T)

Transmit the contents of the Real-Time Clock to bits 16 through 63 of register T. Bits 00 through 15 are cleared. R and S must be zero.

3.2.1.59 3A A 64 NT TRANSMIT (R) TO JOB INTEVAL TIMER

When executed in Job Mode, this instruction transmits bits 32 through 63 of 64-bit register R to the Job Interval Timer. S and T must be zero. (See Sections 3.1.6.3 and 3.1.8.3).

When executed in Monitor Mode, this instruction performs as a no-op.

3.2.1.60 3B A 64 BR DATA FLAG REGISTER LOAD/STORE

Transfer the contents of register R to the data flag register and the original contents of the data flag register to register T. The transfer to and from the data flag register will not occur until all outstanding operations that affect the data flags are

(continued)

3.2.1.60 (Cont'd)

completed. This is not true for the job interval timer and breakpoint inputs. The S designator is undefined and must be set to zero. The R and T designators may be the same and this will swap data flag packages.

NOTE: An immediate data flag branch results at the termination of this instruction if the new contents of the data flag register meet the appropriate conditions.

3.2.1.61 3C 4 32 NT HALF WORD INDEX MULTIPLY(R)*(S) TO (T)

The right-most 24 bits of register R and S contain signed, two's complement integers. Their product is formed and stored into the right-most 24 bits of register T. The left-most 8 bits of register T are cleared to zeros.

If the product or either operand exceeds the value, ±((2**23)-1) the result is undefined.

3.2.1.62 3D 4 64 NT INDEX MULTIPLY (R)*(S) TO (T)

The right-most 48 bits of registers R and S contain signed, two's complement integers. Their product is formed and stored into the right-most 48 bits of register T. The left-most 16 bits of register T are cleared to zeros.

If the product of either operand exceeds the value, ±((2**47)-1) the result is undefined.

3.2.1.63 3E 6 64 IN ENTER(R) WITH I (16 BITS)

Clear register R and transfer the right-most 16 bits of this instruction to the right-most 48 bits of register R (the sign of the 16-bit immediate operand is extended through bit 16).

3.2.1.64 3F 6 64 IN INCREASE(R) BY I (16 BITS)

Replace the right-most 48 bits of register R by the sum of those bits and the right-most 16 bits of this instruction (the sign of the 16-bit immediate operand

(continued)

3.2.1.64 (Cont'd)

Is extended through bit 16 for the addition). Arithmetic overflow is ignored.

3.2.1.65	40	4	32	RG	ADD U; (R)+(S) TO (T)
3.2.1.66	41	4	32	RG	ADD L; (R)+(S) TO (T)
3.2.1.67	42	4	32	RG	ADD N; (R)+(S) TO (T)
3.2.1.68	43				ILLEGAL
3.2.1.69	44	4	32	RG	SUB U; (R)-(S) TO (T)
3.2.1.70	45	4	32	RG	SUB L; (R)-(S) TO (T)
3.2.1.71	46	4	32	RG	SUB N; (R)-(S) TO (T)
3.2.1.72	47				ILLEGAL
3.2.1.73	48	4	32	RG	MPY U; (R)*(S) TO (T)
3.2.1.74	49	4	32	RG	MPY L; (R)*(S) TO (T)
3.2.1.75	4A				ILLEGAL
3.2.1.76	4B	4	32	RG	MPY S; (R)*(S) TO (T)
3.2.1.77	4C	4	32	RG	DIV U; (R)/(S) TO (T)

These instructions perform the indicated floating point arithmetic operation on the 32-bit floating point operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

U signifies that the Upper Result of the opeation is returned; L signifies the Lower Result; S signifies the Significant Result; and N signifies the Normalized Upper Result.

Data flags: bits 41, 42, 43 and 46

3.2.1.78 40 6 32 IN HALF WORD ENTER R WITH I(16 BITS)

Clear register R and transfer the right-most 16 bits of this instruction to the right-most 24 bits of register R (the sign of the 16-bit immediate operand is extended through bit 8).

3.2.1.79 4E 6 32 IN HALF WORD INCREASE R BY I(16 BITS)

Replace the right-most 24 bits of register R by the sum of those bits and the right-most 16 bits of this instruction (the sign of the 16-bit immediate operand is extended through bit 8 for the addition). Arithmetic overflow is ignored.

3.2.1.80 4F 4 32 RG DIV S; (R)/(S) TO (T)

This instruction performs a Divide Significant operation on the 32-bit floating point operands contained in the registers designated by R and S. The result is stored in the register designated by T.

Data flags: bits 41, 42, 43 and 46

3.2.1.81 50 A 32 RG TRUNCATE; (R) TO (T)

Transmit to destination register T the nearest integer whose magnitude is less than or equal to the 32-bit floating point operand in origin register R. This integer is represented as an unnormalized 32-bit floating point number having a positive exponent.

If the exponent of the source operand is positive (greater than or equal to zero), the operand is transmitted directly to the destination register.

If the exponent of the source operand is negative, the magnitude of the coefficient is shifted right end off, and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Zeros are extended on the left during the shift. If the coefficient of the source operand is positive, the shifted coefficient with zero exponent is entered into the destination register. If the coefficient of the source operand is negative, the two's complement of the shifted coefficient with zero exponent is entered into the destination register.

If machine zero is used as an operand, 32 zeros are returned as a result.

Data flag: bit 46

3.2.1.82 51 A 32 RG FLOOR; (R) TO (T)

Transmit to destination register T the nearest integer less than or equal to the 32-bit floating point operand in origin register R. This integer is represented as an unnormalized 32-bit floating point number having a positive exponent.

If the source operand's exponent is positive (greater than or equal to zero), the operand is transmitted directly to the destination register.

(continued)

3.2.1.82 (Cont'd)

If the exponent of the source operand is negative, the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The shifted coefficient with zero exponent is entered into the destination register.

If machine zero is used as an operand, 32 zeros are returned as a result.

Data flag: bit 46

3.2.1.83 52 A 32 RG CEILING;(R) TO (T)

Transmit to destination register T the nearest integer greater than or equal to the 32-bit floating point operand in origin register R. This integer is represented as an unnormalized 32-bit floating point number having a positive exponent.

If the source operand's exponent is positive (greater than or equal to zero), the operand is transmitted directly to the destination register.

If the exponent of the source operand is negative, the two's complement of the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The two's complement of the shifted coefficient with zero exponent is entered into the destination register.

If machine zero is used as an operand, 32 zeros are returned as a result.

Data flag: bit 46

3.2.1.84 53 A 32 RG SIGNIFICANT SQUARE ROOT; (R) TO (T)

Transmit to 32-bit register T the square root of a 32-bit floating point operand in register R.

Data flags: bits 43, 45 and 46

3.2.1.85 54 4 32 RG ADJUST SIGNIFICANCE; (R) PER (S) TO (T)

Adjust the significance of the floating point operand in register R and transmit it to result register T.

A signed, two's complement, integer is contained in the right-most 24 bits of register S. The absolute value of this integer is a shift count.

If the shift count is positive, shift the operand's coefficient left the number of places specified by the shift count or by the number of shifts needed to normalize the coefficient, whichever is smaller. In either case, the exponent of the operand is reduced by one for each place actually shifted. An all zero coefficient will be shifted left the number of places specified.

If the shift count is negative, shift the operand's coefficient right the number of places specified by the shift count and increase the exponent of the operand by one for each place shifted. If R is indefinite, T will be indefinite and data flag bit 46 is set. If R is machine zero, T will be machine zero and data flag bit 43 will be set.

This instruction is undefined if the absolute value of the shift count is greater than 23 base 10. Note that the addition of the shift count can cause either exponent overflow or exponent underflow.

Data flags: bits 42, 43 and 46

3.2.1.86 55 4 32 RG ADJUST EXPONENT; (R) PER (S) TO (T)

Transmit the adjusted operand from register R to result register T. The exponent of the result is set equal to the exponent of the operand in register S. The coefficient of the result is formed by shifting the coefficient of the operand from register R.

The shift count used is the difference between the exponents in registers R and S. If the exponent in register R is greater/less than the exponent in register S, the shift is to the left/right, respectively. For zero coefficients in register R, the exponent from register S is copied to register T with an all-zero coefficient.

(continued)

3.2.1.86 (Cont'd)

If a left shift exceeds the number of places required for normalization, the result is set to indefinite, and data flag bit 42 is set. If either or both operands are indefinite or machine zero, the result is set to indefinite. In this case, data flag bit 46 is set and data flag bit 42 is not set.

Data flags: bits 42 and 46

3.2.1.87 56 7 32 NT SELECT LINK

For certain vector operations (See Table 1), this instruction provides the ability to combine two vector operations into one single operation. The link instruction accomplishes this by chaining the output of the first vector's function to one of the inputs for the second vector's function. The link instruction must be followed immediately by the two vector instructions to be linked such as:

56	R	S	T					
F1	G1	X1	A1	Y1	B1	Z1	C1	(C+1)1
F2	G2	X2	A2	Y2	B2	Z2	C2	(C+1)2

The entire operation will be done as one vector with function F1 preceding function F2. Designators Z2, C2, (C+1)2 and G2 bits 1, 2 will be used to specify the output stream and designators Z1, C1, (C+1)1 and G1 bits 1, 2 will be ignored. Between the two vectors there can only be two input streams (one A and one B) with one broadcast value or one input stream with two broadcast values (one A and one B). Which streams and which broadcast values are selected are specified by G1 bits 3, 4; G2 bits 3, 4 of the vector instructions and R bits 3, 4 of the link instruction. See Table 2 for possible combinations.

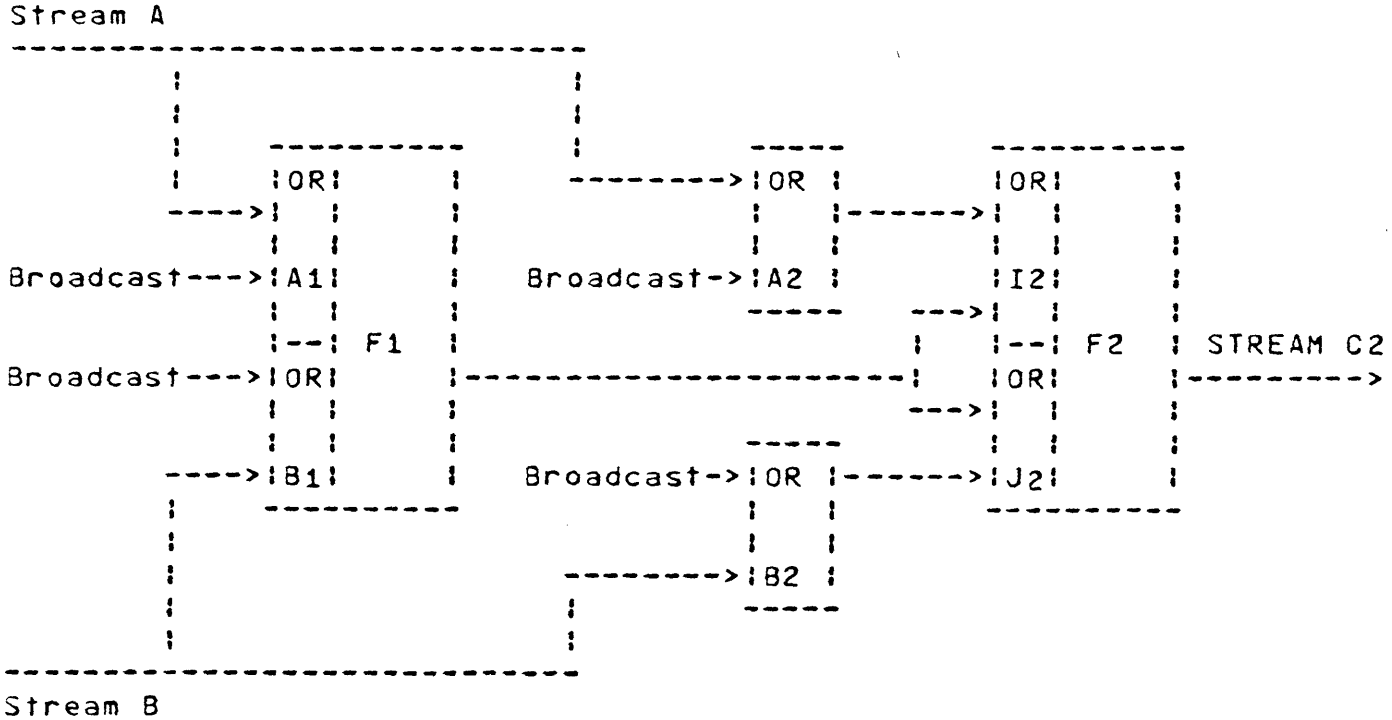
The two inputs to the first function F1 are A1 (selected by designators X1, A1 and G1 bit 3) and B1 (selected by designators Y1, B1 and G1 bit 4). The two inputs (I2 and J2) to the second function (F2) are the output of F1 and either input A2 (selected by designators X2, A2 and G2 bit 3) or input B2

(continued)

3.2.1.87 (Cont'd)

(selected by designators Y2, B2 and G2 bit 4). The input configuration to F2 is determined by R bits 3 and 4 of the link instruction.

R bits	State	Interpretations
	00	Pass
3,4	01	Select A2->I2, F1->J2 and ignore B2
	10	Select B2->J2, F1->I2 and ignore A2
	11	Select F1->I2 and F1->J2 and ignore B2 & A2
0-2 5-7		Undefined and must be set to zero



(continued)

3.2.1.87 (Cont'd)

- o G1 bit 0 and G2 bit 0 must be equal.
- o G1 bits 5-7 apply to F1.
- o G2 bits 5-7 apply to F2.
- o S and T designators of link are undefined and must be set to zero.

TABLE 1

Instructions that can be used in a link operation.

FIRST VECTOR (F1)		SECOND VECTOR (F2)	
*	Instruction	*	Instruction
1	8A	1	8A
2	9D	2	9D
3	88,89,8B	3	88,89,8B
4	80,81,82,83,84,85,86,87,90,91,92	4	80,81,82,83,84,85,86,87,90,91,92,C4,C5,C6,C7

* Functional unit number where:

1. Array shift
2. Pack, Exponents, Array Logical
3. Array Multiply
4. Array Add

- o The operation is undefined if the instructions selected for F1 and F2 have the same functional unit number.

(continued)

3.2.1.87 (Cont'd)

TABLE 2

Combinations of R, G1, and G2 bits 3 and 4 that can be selected.

	R Bits 3,4=01					R Bits 3,4=10					R Bits 3,4=11				
G1 Bit 3 =	0	1	1	1	0	0	0	0	1	1	0	0	1	1	
G1 Bit 4 =	0	0	0	1	1	0	1	1	1	0	0	1	0	1	
G2 Bit 3 =	1	0	1	0	1	0	0	0	0	0	0	0	0	0	
G2 Bit 4 =	0	0	0	0	0	1	0	1	0	1	0	0	0	0	

- Combinations other than above produce undefined results.

3.2.1.88 57 ILLEGAL

3.2.1.89 58 A 32 RG TRANSMIT; (R) TO (T)

Transmit the operand in 32-bit register R to 32-bit register T.

3.2.1.90 59 A 32 RG ABSOLUTE; (R) TO (T)

Transmit the absolute value of the 32-bit floating point operand in register R to register T.

3.2.1.91 5A A 32 RG EXP.; (R) TO (T)

Transmit the exponent from the left-most 8-bit positions of the origin register R to the right-most 8-bit positions of destination register T. The sign of the exponent is extended through bit 8 of destination register T, the left-most 8 bits of the destination register are cleared to zeros.

3.2.1.92 5B 4 32 RG PACK; (R), (S) TO (T)

Transmit a 32-bit floating point number to the destination register T. The exponent of the number is obtained from the right-most 8-bit positions of register R and the coefficient is obtained from the right-most 24-bit positions of register S.

3.2.1.93 5C A B RG EXTEND; 32 BIT(R) TO 64 BIT(T)

Extend the floating point number from 32-bit register R into a 64-bit floating point number and transmit the result to 64-bit register T. The value of the resulting 16-bit exponent is 24 less than that of the origin operand's exponent. The coefficient is obtained by transmitting the right-most 24 bits of the origin register into bits 16 through 39 of the destination register. The right-most 24 bits of the destination register are cleared to zero.

If R is indefinite, T will be indefinite and data flag bit 46 will be set. If R is machine zero, T will be machine zero and data flag bit 43 will be set.

Data flag: bit 43 and 46

3.2.1.94 5D A B RG INDEX EXTEND; 32 BIT(R) TO 64 BIT(T)

Extend the floating point number from 32-bit register R into a 64-bit floating point number and transmit the result to 64-bit register T. The value of the resulting 16-bit exponent is the same as the origin operand's exponent. The coefficient is obtained by transmitting the right-most 24 bits of the origin register into bits 40 through 63 of the destination register. Bits 16 through 39 of the destination register are set to the sign of the origin coefficient.

If R is indefinite, T will be indefinite and data flag bit 46 will be set. If R is machine zero, T will be machine zero and data flag bit 43 will be set.

Data flag: bit 43 and 46

3.2.1.95 5E 7 32 NT LOAD; (T) PER (S), (R)
3.2.1.96 5F 7 32 NT STORE; (F) PER (S), (R)

Load/store 32-bit register T from/into the address specified by (R) + (S) where (R) is the base address and (S) is an item count of half-words. Note that S and R are 64-bit registers and that the item count is shifted left five places before the addition. Overflow from this addition is ignored, if it occurs.

------- SUPER COMPUTER OPERATIONS --------

3.2.1.97 60 4 64 RG ADD U; (R)+(S) TO (T)
3.2.1.98 61 4 64 RG ADD L; (R)+(S) TO (T)
3.2.1.99 62 4 64 RG ADD N; (R)+(S) TO (T)

These instructions perform the indicated floating point arithmetic operation on the 64-bit floating point operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

U signifies that the upper result of the operation is returned; L signifies the lower result; and N signifies the normalized upper result.

Data flags: bits 42, 43 and 46

3.2.1.100 63 4 64 RG ADD ADDRESS; (R)+(S) TO (T)

This instruction adds bits 16 through 63 of register R to bits 16 through 63 of register S and stores the result in bits 16 through 63 of register T. Bits 16 through 63 are treated as 48-bit, positive, unsigned integers. Arithmetic overflow is ignored. Bits 0 through 15 of register R are transferred without modification to bits 0 through 15 of register T.

3.2.1.101 64 4 64 RG SUB U; (R)-(S) TO (T)
3.2.1.102 65 4 64 RG SUB L; (R)-(S) TO (T)
3.2.1.103 66 4 64 RG SUB N; (R)-(S) TO (T)

These instructions perform the indicated floating point arithmetic operation on the 64-bit floating point operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

U signifies that the upper result of the operation is returned; L signifies the lower result; and N signifies the normalized upper result.

Data flags: bits 42, 43 an 46

3.2.1.104 67 4 64 RG SUB ADDRESS; (R)-(S) TO (T)

This instruction subtracts bits 16 through 63 of register S from bits 16 through 63 of register R and stores the result in bits 16 through 63 of register T. Bits 16 through 63 are treated as 48-bit, positive unsigned integers. Arithmetic overflow is ignored. Bits 0 through 15 of register R are transferred without modification to bits 0 through 15 of register T.

3.2.1.105	68	4	64	RG	MPY U; (R)*(S) TO (T)
3.2.1.106	69	4	64	RG	MPY L; (R)*(S) TO (T)
3.2.1.107	6A				ILLEGAL
3.2.1.108	68	4	64	RG	MPY S; (R)*(S) TO (T)
3.2.1.109	6C	4	64	RG	DIV U; (R)/(S) TO (T)

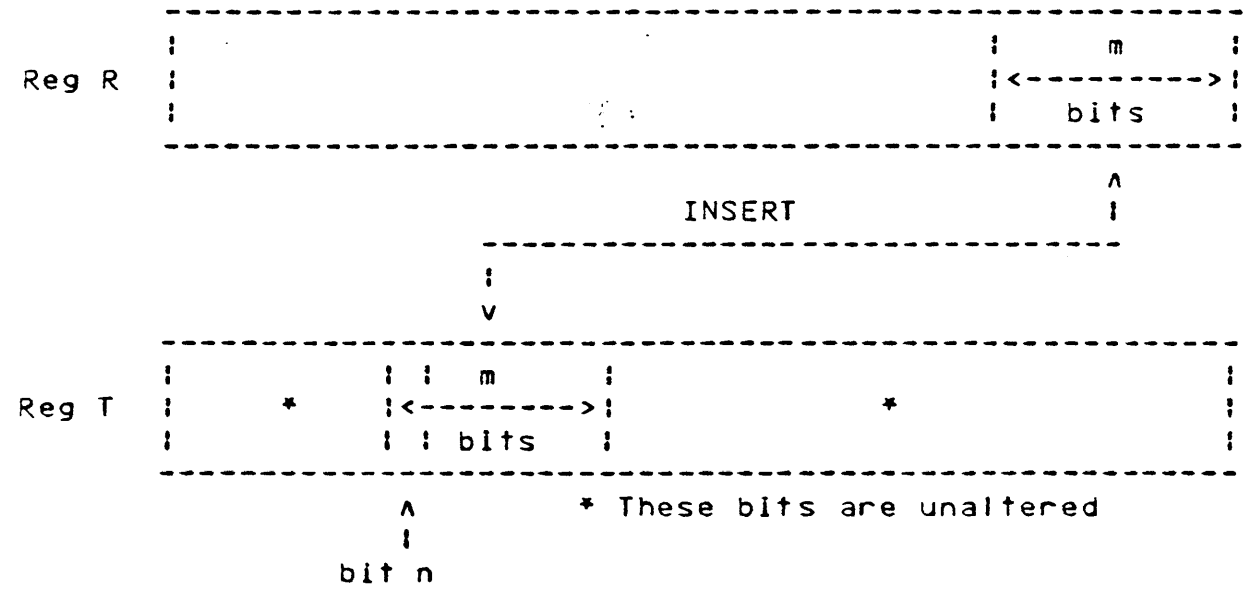
These instructions perform the indicated floating point arithmetic operation on the 64-bit floating point operands contained in the registers designated by R and S. The result in each case is stored in the register designated by T.

U signifies that the upper result of the operation is returned; L signifies the lower result; S signifies the significant result.

Data flags: bits 41, 42, 43 and 46

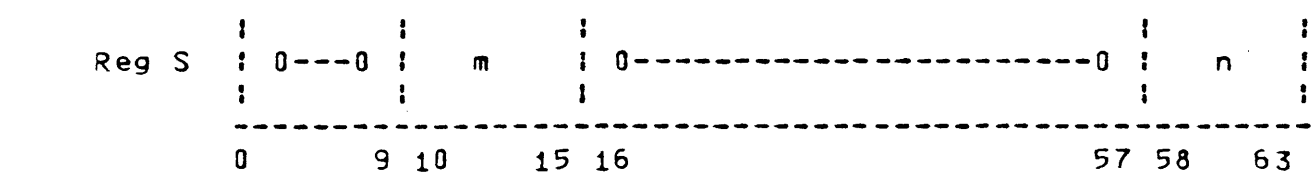
3.2.1.110	60	4	64	RG	INSERT BITS; (R) TO (T) PER (S)

This instruction inserts the right-most bits of the register designated by R into the register designated by T.



(continued)

3.2.1.110 (Cont'd)

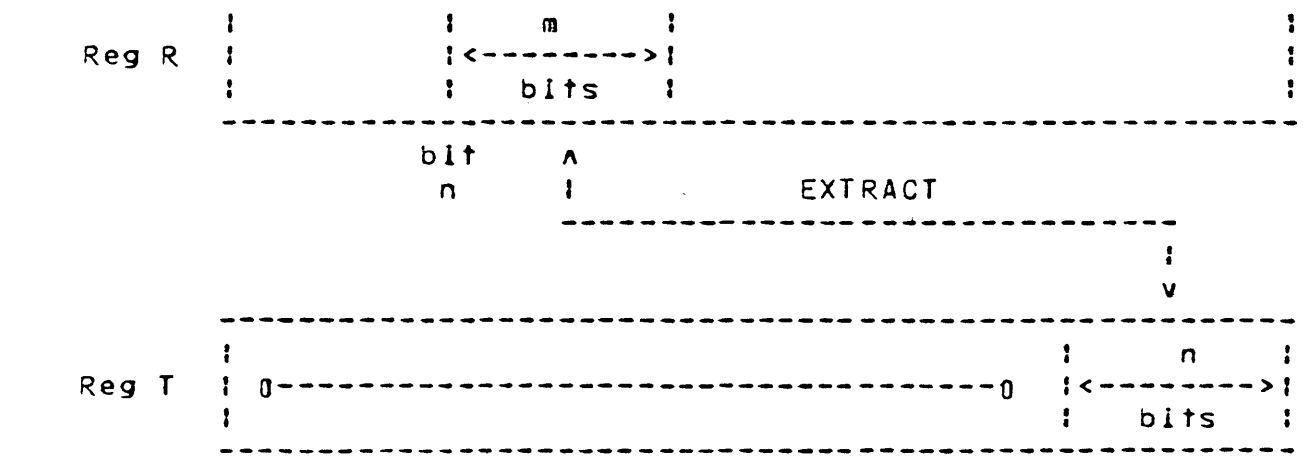


Bits 10 through 15 of the register specified by S contain the number (m) of right-most bits to be inserted. The right-most 6 bits of register S specify the the bit number (n) in register T where the left-most bit of the inserted data will be placed. Bits 0 through 9 and 16 through 57 of register S are undefined and must be set to zero.

If the R designator is equal to zero, then register zero will provide machine zero. If m plus n is greater than 64[D], or if m is equal to zero, the results of this instruction are undefined.

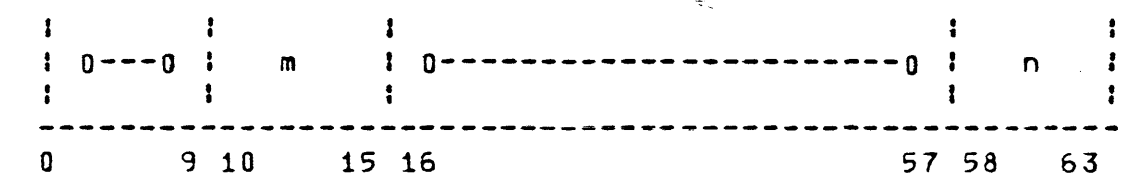
3.2.1.111 6E 4 64 RG EXTRACT BITS; (R) TO (T) PER (S)

This instruction extracts bits from the register designated by R and stores them into the right-most portion of the register specified by T. Register T is cleared before receiving the extracted bits.



(continued)

3.2.1.111 (Cont'd)



Bits 10 through 15 of the register specified by S contain the number (m) of bits to be extracted from register R. The right-most 6 bits of register S specify the left-most bit number of the extracted bits. Bits 0 through 9 and 16 through 57 of register S are undefined and must be set to zero.

If the R designator is equal to zero, register zero will provide machine zero. If m plus n is greater than 64[D], or if m is equal to zero, the results of this instruction are undefined.

3.2.1.112 6F 4 64 RG DIV S; (R)/(S) TO (T)

This instruction performs a Divide Significant operation on the 64-bit floating point operands contained in the registers designated by R and S. The result is stored in the register designated by T.

Data flags: bits 41, 42, 43 and 46

3.2.1.113 70 A 64 RG TRUNCATE; (R) TO (T)

Transmit to destination register T the nearest integer whose magnitude is less than or equal to the magnitude of the 64-bit floating point operand in origin register R. The integer is represented as an unnormalized 64-bit floating point number having a positive exponent.

If the exponent of the source operand is positive (greater than or equal to zero), the operand is transmitted directly to the destination register.

If the exponent of the source operand is negative, the magnitude of the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Zeros are extended on the left during the shift. If the coefficient of the source operand is positive,

(continued)

3.2.1.13 (Cont'd)

the shifted coefficient with zero exponent is entered into the destination register. If the coefficient of the source operand is negative, the two's complement of the shifted coefficient with zero exponent is entered into the destination register.

If a machine zero is used as an operand, 64 zeros are returned as a result.

Data flag: bit 46

3.2.1.114 71 A 64 RG FLOOR; (R) TO (T)

Transmit to destination register T the nearest integer less than or equal to the 64-bit floating point operand in origin register R. This integer is represented as an unnormalized 64-bit floating point number having a positive exponent.

If the source operand's exponent is positive (greater than or equal to zero), the operand is transmitted directly to the destination register.

If the exponent of the source operand is negative, the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The shifted coefficient with zero exponent is entered into the destination register.

If a machine zero is used as an operand, 64 zeros are returned as a result.

Data flag: bit 46

3.2.1.115 72 A 64 RG CEILING; (R) TO (T)

Transmit to destination register T the nearest integer greater than or equal to the 64-bit floating point operand in origin register R. This integer is represented as an unnormalized 64-bit floating point number having a positive exponent.

If the source operand's exponent is positive (greater than or equal to zero), the operand is transmitted directly to the destination register.

(continued)

3.2.1.115 (Cont'd)

If the exponent of the source operand is negative, the two's complement of the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The two's complement of the shifted coefficient with zero exponent is entered into the destination register.

If machine zero is used as an operand, 64 zeros are returned as a result.

Data flag: bit 46

3.2.1.116 73 A 64 RG SIGNIFICANT SQUARE ROOT; (R) TO (T)

Transmit to register T the square root of the 64-bit floating point operand in register R.

Data flags: bits 43, 45 and 46

3.2.1.117 74 4 64 RG ADJUST SIGNIFICANCE; (R) PER (S) TO (T)

Adjust the significance of the floating point operand in register R and transmit it to result register T.

A signed, two's complement integer is contained in the right-most 48 bits of register S. The absolute value of this integer is a shift count. The left-most 16 bits of register S are ignored.

If the shift count is positive, shift the operand's coefficient left the number of places specified by the shift count or by the number of shifts needed to normalize the coefficient, whichever is smaller. In either case, the exponent of the operand is reduced by one for each place actually shifted. An all zero coefficient will be shifted left the number of places specified.

If the shift count is negative, shift the operand's coefficient right the number of places specified by the shift count and increase the exponent of the operand by one for each place shifted.

(continued)

3.2.1.117 (Cont'd)

This instruction is undefined if the absolute value of the shift count is greater than 47[D]. Note that the addition of shift count can cause either exponent overflow or exponent underflow.

If R is indefinite, T will be definite and data flag bit 46 will be set. If R is machine zero, T will be machine zero and data flag bit 43 will be set.

Data flags: bits 42, 43 and 46

3.2.1.118 75 4 64 RG ADJUST EXPONENT; (R) PER (S) TO (T)

Transmit the adjusted operand from register R to result register T. The exponent of the result is set equal to the exponent of the operand in register S. The result is formed by shifting the coefficient of the operand from register R.

The shift count used is the difference between the exponents is register R and S. If the exponent in register R is greater/less than the exponent in register S, the shift is to the left/right, respectively. For zero coefficients in register R, the exponent from register S is copied to register T with an all-zero coefficient.

If a left shift exceeds the number of places required for normalization, the result is set to indefinite and data flag 42 is set. If either or both operands are indefinite or machine zero, the result is set to indefinite. In this case, data flag bit 46 is set and data flag bit 42 is not set.

3.2.1.119 76 A B RG CONTRACT; 64 BIT (R) TO 32 BIT (T)

Contract the 64-bit floating point number from register R into a 32-bit floating point number and transmit the result to 32-bit register T.

(continued)

3.2.1.119 (Cont'd)

Input Exponent	Result
7FFF : 7000	Result Indefinite Indefinite Data Flag 46
6FFF : 0058	Result Indefinite Data Flag 42, 46
0057 : : : FF78	Result exponent 24[D] larger than input exponent Copy left-most 24 bits of input coefficient
FF77 : 8000	Result machine zero Data Flag 43

The 24-bit result coefficient is copied from the left-most 24 bits of the 48-bit source coefficient (bits 16 through 39). This has the effect of contracting all negative source coefficients, whose absolute values (neglecting the exponent) were less than or equal to (2**24), to a minus one.

Data flags: bits 42, 43 and 46

3.2.1.120 77 A B RG ROUNDED CONTRACT; 64 BIT (R) TO 32 BIT (T)

Perform a rounded contract operation on the 64-bit floating point number in register R and transmit the 32-bit floating point result to 32-bit register T. A positive one is added to the origin operand in bit position 40. If overflow occurs the exponent is increased by one and the coefficient is shifted right one place. The left-most 24 bits of this 48-bit sum

(continued)

3.2.1.120 (Cont'd)

are then transmitted to the 24-bit coefficient portion of register T. Each non-endcase result element's 8-bit exponent is 24[D] (25[D] if overflow occurred) greater than the corresponding source element's exponent.

Data flags: bits 42, 43 and 46

3.2.1.121 78 A 64 RG TRANSMIT; (R) TO (T)

Transmit the 64-bit operand in register R to register T.

3.2.1.122 79 A 64 RG ABSOLUTE; (R) TO (T)

Transmit the absolute value of the 64-bit floating point operand in register R to register T.

Data flags: bits 42, 43 and 46

3.2.1.123 7A A 64 RG EXP.; (R) TO (T)

Transmit the exponent from the left-most 16-bit positions of origin register R to the right-most 16-bit positions of destination register T. The sign of the exponent is extended through bit 16 of destination register T. The left-most 16 bits of the destination register are cleared to zeros.

3.2.1.124 7B 4 64 RG PACK; (R), (S) TO (T)

Transmit a 64-bit floating point number to destination register T. The exponent of the number is obtained from the right-most 16-bit positions of register R, and the coefficient is obtained from the right-most 48-bit positions of register S.

3.2.1.125 7C A 64 RG LENGTH; (R) TO (T)

Transmit the left-most 16-bit positions of origin register R to the right-most 16-bit positions of destination register T. The left-most 48 bits of the destination register are cleared to zeros.

3.2.1.126 7D 7 64 NT SWAP; S-----\>T AND R-----\>S

Move to destination field T, a portion of the register file beginning at the 64-bit register specified by the right-most eight bits of register S. Transmit source field R to the register file beginning at the 64-bit register specified by the right-most eight bits of register S.

The left-most 16 bits of register R and T specify the field length in words for the source and destination fields, respectively. The field lengths of the source and destination fields may be different but each must be even. A zero field length indicates no transfer for that field. Any transfer of words into or out of the register file that becomes exhausted of registers (i.e., beyond the bounds of the register file), causes the instruction to become undefined.

The right-most 48 bits of registers R and T specify the base address of the source and destination fields, respectively. These addresses must specify an even 64-bit word in central storage. Bits 57 through 63 of register R and T are undefined and must be set to zero. Overlap of the source and destination fields is allowed only if the base addresses for both fields are equal.

Registers R, S, or T, may be in the range of the registers being swapped.

The starting register in the file specified by the right-most eight bits of the register specified by S must be an even register or this instruction will be treated as an undefined instruction. For additional material see Section 3.1.7 on the Register File.

3.2.1.127 7E 7 64 NT LOAD; (T) PER (S), (R)
3.2.1.128 7F 7 64 NT STORE; (T) PER (S), (R)

Load/store 64-bit register T from/into the address specified by (R) + (S) where (R) is the base address and (S) is an item count of words.

3.2.1.129 80 1 E VT ADD U; A+B----\>C
3.2.1.130 81 1 E VT ADD L; A+B----\>C

3.2.1.131 82 1 E VT ADD N; A+B---->C

These instructions perform the indicated floating point arithmetic operations on elements of vectors A and B. The results are stored into vector C.

U Signifies that the upper result of the operation is returned; L signifies the lower result and N signifies the normalized upper result. Sign Control is permitted, see 3.1.4.9 for details.

Data flags: bits 42, 43 and 46

3.2.1.132 83 1 64 VT ADD ADDRESS; A+B---->C

This instruction adds bits 16 through 63 of the elements of the B vector to bits 16 through 63 of the elements of the A vector and stores the results in bits 16 through 63 of the elements of the C vector. Bits 16 through 63 are treated as 48 bit, positive, unsigned integers. Arithmetic overflow is ignored. Bits 0 through 15 of the elements of the A vector are transferred without modification to bits 0 through 15 of the elements of the C vector. Bit 0, 5, 6, and 7 of the G designators must be set to zero.

3.2.1.133 84 1 E VT SUB U; A-B---->C
3.2.1.134 85 1 E VT SUB L; A-B---->C
3.2.1.135 86 1 E VT SUB N; A-B---->C

These operations perform the indicated floating point arithemtic operations on elements of vectors A and B. The results are stored into vector C.

U signifies that the upper result of the operation is returned; L signifies the lower result; and N signifies the normalized upper result. Sign Control is permitted, see 3.1.4.9 for details.

Data flags: bits 42, 43 and 46

3.2.1.136 87 1 64 VT SUB ADDRESS; A-B---->C

This instruction subtracts bits 16 through 63 of the elements of the B vector from bits 16 through 63 of the elements of the A vector and stores the results in bits 16 through 63 of the elements of the C vector. Bits 16 through 63 are treated as 48 bit, positive, unsigned integers. Arithmetic overflow is ignored. Bits 0 through 15 of the elements of the

A vector are transferred without modification to bits 0 through 15 of the elements of the C vector. Bit 0, 5, 6 and 7 of the G designator must be set to zero.

3.2.1.137 88 1 E VT MPY U; A*B---->C
3.2.1.138 89 1 E VT MPY L; A*B---->C

These instructions perform the indicated floating point arithmetic operations on elements of vectors A and B. The results are stored into vector C.

U signifies that the upper result of the operation is returned; L signifies the lower result; and S signifies the significant result. Sign Control is permitted, see 3.1.4.9 for details.

Data flags: bits 41, 42, 43 and 46

3.2.1.139 8A 1 64 VT SHIFT; A PER B----> C

This instruction shifts the 64-bit elements from source vector A by corresponding elements from source vector B and stores them into result vector C. If the rightmost byte of the element in vector B is in the range from 0 through 3F base 16 (0 trhough 63 base 10), the element from vector A is shifted left end-around the number of specified places. If the rightmost byte of the element in vector B is in the range from FF through C1 base 16 (-1 through -63 base 10), the element from vector A is shifted right with sign extension. Bit 0 of operands in vector A is the sign bit for extension and the number for right shifts is equal to the two's complement of the rightmost bytes of operands in vector B. If the rightmost byte of elements from vector B is greater than 3F or less than C1 base 16, the results are undefined. The leftmost seven bytes of elements in vector B are ignored.

(continued)

3.2.1.139 (Cont'd)

G bits 0 and 5-7 are undefined and must be set to zero.

3.2.1.140	8B	1	E	VT	MPY S; A*B----->C
3.2.1.141	8C	1	E	VT	DIV U; A/B----->C
3.2.1.142	8D				ILLEGAL
3.2.1.143	8E				ILLEGAL
3.2.1.144	8F	1	E	VT	DIV S; A/B----->C

These instructions perform the indicated floating point arithmetic operations on elements of vectors A and B. The results are stored into vector C.

U signifies that the upper result of the operation is returned; L signifies the lower result; and S signifies the significant result. Sign Control is permitted, see 3.1.4.9 for details.

Data flags: bits 41, 42, 43 and 46

3.2.1.145	90	1	E	VT	TRUNCATE; A----->C

Each element of result vector C is the nearest integer whose magnitude is less than or equal to the magnitude of the corresponding floating point element of source vector A. This integer is represented by an unnormalized floating point number having a positive exponent.

If the source element's exponent is positive (greater than or equal to zero), the element is transmitted directly to the result field.

If the exponent of the source element is negative, the magnitude of the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Zeros are extended on the left during the shift. If the coefficient of the source element is positive, the shifted coefficient with zero exponent is transferred to the result field. If the coefficient of the source element is negative, the two's complement of the shifted coefficient with zero exponent is transferred to the result field.

The Y and B designators and bits 4-7 of the G designator are undefined and must be set to zeros.

(continued)

3.2.1.145 (Cont'd)

If machine zero is used as an operand element, the result element will be all zero.

Data flag: Bit 46

3.2.1.146 91 1 E VT FLOOR;A--->C

Each element of result vector C is the nearest integer less than or equal to the corresponding floating point element of source vector A. This integer is represented by an unnormalized floating point number having a positive exponent.

If the source element's exponent is positive (greater than or equal to zero), the element is transmitted directly to the result field.

If the exponent of the source element is negative, the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The shifted coefficient with zero exponent is transferred to the result field.

The Y and B designators and bits 4-7 of the G designator are undefined and must be set to zeros.

If machine zero is used as an operand element, the resulting element will be all zero

Data flag: bit 46

3.2.1.147 92 1 E VT CEILING;A--->C

Each element of result vector C is the nearest integer greater than or equal to the corresponding floating point element of source vector A. This integer is represented by an unnormalized floating point number having a positive exponent.

If the source element's exponent is positive (greater than or equal to zero), the element is transmitted directly to the result field.

(continued)

3.2.1.147 (Cont'd)

If the exponent of the source element is negative, the two's complement of the coefficient is shifted right end off and the exponent is increased by one for each bit position shifted until the exponent becomes zero. Sign bits are extended on the left during the shift. The two's complement of the shifted coefficient with zero exponent is transferred to the result field.

The Y and B designators and bits 4-7 of the G designator are undefined and must be set to zeros.

If machine zero is used as an operand element, the resulting element will be all zero.

Data flag: bit 46

3.2.1.148 93 1 E VT SIGNIFICANT SQUARE ROOT; A--->C

This instruction forms the square root of each element of vector A and places the result in vector C.

The Y and B designators and bits 4 and 7 of the G designator are undefined and must be set to zero. Sign Control is permitted, see 3.1.4.9 for details.

Data flag: bits 43, 45 and 46

3.2.1.149 94 1 E VT ADJUST SIGNIFICANCE; A PER B--->C

Adjust the significance of the floating point elements from vector A and transmit them to result vector C.

Signed, two's complement integers are contained in the rightmost 48 (24) bits of the elements from vector B. The absolute value of these integers are shift counts.

If a shift count is positive, shift the coefficient from vector A left the number of places specified by the shift count or by the number of shifts needed to normalize the coefficient, whichever is smaller. In either case, the exponent of the element is reduced

(continued)

3.2.1.149 (Cont'd)

by one for each place shifted. An all zero coefficient will be shifted left the number of places specified.

If the shift count is negative, shift the element's coefficient right the number of places specified by the shift count and increase the exponent of the element by one for each place shifted.

The result stored in vector C is undefined if the absolute value of the shift count is greater than 47[D] (23[D] for 32-bit operands). Note that the addition of the shift count to the exponent can cause either exponent overflow or exponent underflow.

If A is indefinite, C will be indefinite and data flag bit 46 is set. If A is machine zero, C will be machine zero and data flag bit 43 will be set.

Bits 5-7 of the G designator are undefined and must be set to zeros.

Data flags: bits 42, 43 and 46

3.2.1.150 95 1 E VT ADJUST EXPONENT; A PER B--->C

Transmit adjusted elements from vector A to result vector C. The exponent of a result element is set equal to the exponent of the associated element from vector B. The coefficient of the result elements are formed by shifting the coefficients of the operand elements from vector A.

The shift count used is the difference between the exponents of associated elements from A and B. If the exponent of the element from A is greater/less than the exponent of the element from B, the shift is to the left/right, respectively. For zero coefficients in vector A, the exponent from vector B is copied to vector C with an all-zero coefficient.

If a left shift exceeds the number of places required for normalization, the result is set to indefinite, and data flag bit 42 is set. If either or both operands are indefinite or machine zero, the result is set to indefinite. In this case, data flag bit 46 is set and data flag bit 42 is not set.

Bits 5-7 of the G designator are undefined and must be set to zeros.

Data flags: bits 42 and 46

3.2.1.151 96 1 B VT CONTRACT; 64 BIT A--->32 BIT C

Each 32-bit floating point element of result vector C is formed by contracting the corresponding 64-bit floating point element of source vector A. Each non-endcase 8-bit result element's exponent is 24[D] greater than its source element's exponent. See the 76 Instruction (Register Contract) for detail.

Each 24-bit result coefficient is copied from the left-most 24 bits of its 48-bit source coefficient (bits 16 through 39). This has the effect of contracting all negative source coefficients, whose absolute values (neglecting the exponent) were less than or equal to (2**24), to a minus one.

The Y and B designators and bits 0, 4, 5, 6 and 7 of the G designator are undefined and must be set to zero.

Data flags: bits 42, 43 and 46.

3.2.1.152 97 1 B VT ROUNDED CONTRACT; 64 BIT A--->32 BIT C

Each 32-bit floating point element of result vector C is formed by performing a rounded contract operation on the corresponding 64-bit floating point element of source vector A. A positive one is added to the origin operand in bit position 40. If overflow occurs, the exponent is increased by one and coefficient is shifted right one place. The left most 24 bits of this 48-bit sum are then transmitted to the 24-bit coefficient portion of the result vector element.

Each non-endcase result element's 8-bit exponent is 24[D] (25[D] if overflow occurred) greater than the corresponding source element's exponent.

The Y and B designators and bits 0, 4, 5, 6 and 7 of the G designators are undefined and must be set to zero.

Data flags: bits 42, 43 and 46

3.2.1.153 98 1 E VT TRANSMIT;A--->C

Transmit source vector A to result vector C.

The Y and B designators and bits 4-7 of the G designator are undefined and must be set to zeros.

3.2.1.154 99 1 E VT ABSOLUTE;A--->C

Each element of result vector C is the absolute value of the corresponding element of vector A. The vectors contain floating point elements.

The Y and B designators and bits 4-7 of the G designator are undefined and must be set to zeros.

Data flag: bits 42, 43 and 46

3.2.1.155 9A 1 E VT EXP.; A--->C

The elements of result vector C are formed by storing the exponents from input vector A into the right-most portion of the coefficients of vector C. The sign of

(continued)

3.2.1.155 (Cont'd)

the exponent is extended left to the coefficient sign bit position. The exponent portion of each element of vector C is cleared to zero.

The Y and B designators and bits 4-7 of the G designator are undefined and must be set to zeros.

3.2.1.156 9B 1 E VT PACK;A, B--->C

Transmit to result vector C a 64/32 bit floating point number produced as follows. The result is formed by transmitting the right-most 16/8 bit positions of an element of source vector A (exponent) to the left-most 16/8 bit positions of result vector C and the right-most 48/24 bit positions of an element of source vector B (coefficient) to the right-most 48/24 bit positions of result vector C.

3.2.1.157 9C 1 B VT EXTEND; 32 BIT A--->64 BIT C

The elements of result vector C are formed by extending the 32-bit floating point operands of vector A into 64-bit floating point operands. The value of each of the resulting 16-bit exponents is 24[D] less than that of the corresponding source element's exponent. The coefficient of each result is obtained by transmitting the right-most 24 bits of the corresponding source element into bits 16 through 39 of the result element. The right-most 24 bits of each result are cleared to zeros.

If bit 3 of the G designator is set, indicating broadcast of the A register, the 8-bit A designator will be interpreted as a 32-bit register designator.

If an element of vector A is indefinite, the corresponding element of vector C will be set to indefinite and data flag bit 46 set. If an element of vector A is machine zero, machine zero will be stored in vector C and data flag bit 43 will be set.

The Y and B designators and bits 0, 4-7 of the G designator are undefined and must be set to zeros.

Data flag: bit 43 and 46

------- S U P E R C O M P U T E R O P E R A T I O N S --------

3.2.1.158 9D 1 E VT LOGICAL; A, B----> C

This instruction performs the bit by bit logical operation selected by G bits 5-7 between source vectors A and B with results stored in vector C.

G567=		000	001	010	011	100	101	110	111
		EXCL. OR	AND	OR	STROKE	PIERCE	IMPLI.	INHIBIT	EQUIV.
					$\overline{(A.B)}$	$\overline{(A+B)}$	$\overline{A}+B$	$A.\overline{B}$	$A-\overline{B}$
A	B	A-B	A.B	A+B	(A.B)	(A+B)	A+B	A.B	A-B
0	0	0	0	0	1	1	1	0	1
0	1	1	0	1	1	0	0	0	0
1	0	1	0	1	1	0	1	1	0
1	1	0	1	1	0	0	1	0	1

3.2.1.159	9E				ILLEGAL
3.2.1.160	9F				ILLEGAL
3.2.1.161	A0	2	E	SV	ADD U; A+B-->C
3.2.1.162	A1	2	E	SV	ADD L; A+B-->C
3.2.1.163	A2	2	E	SV	ADD N; A+B-->C
3.2.1.164	A3				ILLEGAL
3.2.1.165	A4	2	E	SV	SUB U; A-B-->C
3.2.1.166	A5	2	E	SV	SUB L; A-B-->C
3.2.1.167	A6	2	E	SV	SUB N; A-B-->C
3.2.1.168	A7				ILLEGAL
3.2.1.169	A8	2	E	SV	MPY U; A*B--->C
3.2.1.170	A9	2	E	SV	MPY L; A*B--->C
3.2.1.171	AA				ILLEGAL
3.2.1.172	AB	2	E	SV	MPY S; A*B--->C
3.2.1.173	AC	2	E	SV	DIV U; A/B--->C
3.2.1.174	AD				ILLEGAL
3.2.1.175	AE				ILLEGAL
3.2.1.176	AF	2	E	SV	DIV S; A/B--->C

These instructions perform the indicated floating point operation on elements of sparse vectors A and B and return the results to sparse vector C. An element is read from sparse vector A whenever a one bit is encountered in order vector X. An element is read from sparse vector B whenever a one bit is found in order vector Y. A zero bit in source order vector causes machine zero (normalized one for multiplies and divides) to be used as the associated A and/or B element.

(continued)

3.2.1.176 (Cont'd)

Output order vector Z is a bit by bit logical function specified by G bits 1 and 2 as follows:

G bit 1	G bit 2	
0	0	Z equals the logical or of X and Y for add/subtract operations. Z equals the logical and of X and Y for multiply/divide operations
0	1	Z equals the logical and of X and Y for add/subtract operations. Z equals the logical or of X and Y for multiply/divide operations.
1	0	Z equals the logical exclusive or of X and Y for all operations.
1	1	Z equals the logical implication of X and Y for all operations

X	Y	OR	AND	EXCLUSIVE OR	IMPLICATION
0	0	0	0	0	1
0	1	1	0	1	0
1	0	1	0	1	1
1	1	1	1	0	1

An output element of sparse vector C is generated for each 1 bit in order vector Z.

These instructions follow the rules for Sparse Vector Termination.

The resulting length of sparse vector C is transferred to the length specification portion of register C.

U signifies that the Upper Result of the operation is returned; L signifies the Lower Result; N signifies the Normalized Upper Result and S signifies the significant result. Sign Control is permitted, see 3.1.4.9 for details.

(continued)

3.2.1.176 (Cont'd)

NOTE: Data flags are set only for output elements of sparse vector C.

G bits 3 and/or 4 are used to broadcast A and/or B, respectively.

Data flags: bits 41, 42, 43 and 46

Sparse Vector Floating Add Example

	F	G	X	A	Y	B	Z	C
A	0:8	0:0	3:0	4:0	5:0	6:0	7:0	8:

Before Execution

Register 03 = 0007000000003000
04 = 0000000000004000
05 = 0008000000005000
06 = 0000000000006000
07 = 0009000000007000
08 = 0000000000008000

Address 4000 - 4040

A 0	A 1	A 2

3000 - 3006

1	0	0	1	0	0	1

6000 - 60A0

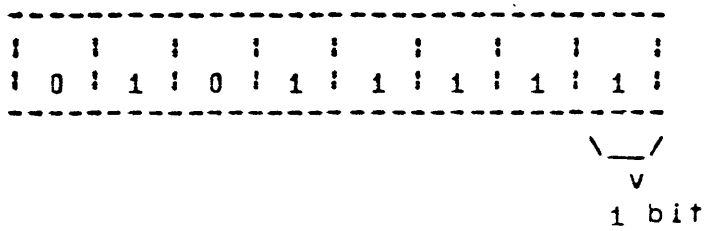
B 0	B 1	B 2	B 3	B 4	B 5

____/
v
32 bits

(continued)

3.2.1.176 (Cont'd)

Address 5000-5007

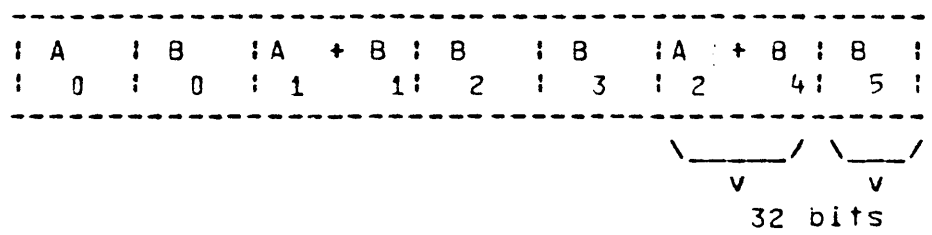


After Execution

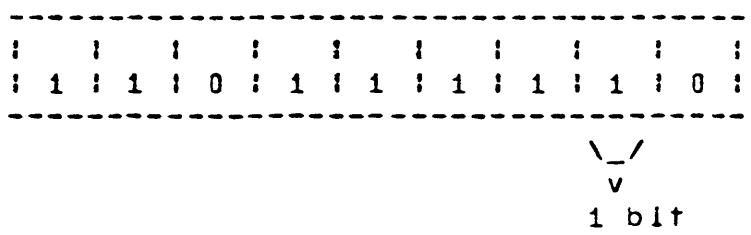
Registers 03, 04, 05, 06 and 07 are unchanged.

Register 08 = 000700000008000

Address 8000 - 80C0



7000 - 7008



(continued)

3.2.1.176 (Cont'd)

Sparse Vector Floating Point Multiply Example:

F	G	X	A	Y	B	Z	C
A	8:8	0:0	4:0	5:0	6:0	7:0	8:0	9:

Before Execution:

Register 04 = 0008000000001000
05 = 0000000000002000
06 = 0008000000003000
07 = 0000000000004000
08 = 0009000000005000
09 = 0000000000006000

Address 2000 - 2060

A 0	A 1	A 2	A 3

1000 - 1007

1	0	0	1	0	0	1	1

4000 - 40A0

B 0	B 1	B 2	B 3	B 4	B 5

___/ v 32 bits

3000 - 3007

0	1	0	1	1	1	1	1

__/ v 1 bit

After execution:

Register 04, 05, 06, 07, and 08 are not changed.

Register 09 = 0003000000006000

Address 6000 - 6040

A * B	A * B	A * B
1 1	2 4	3 5

5000 - 5008

0	0	0	1	0	0	1	1	0

------- S U P E R C O M P U T E R O P E R A T I O N S --------

3.2.1.177	B0	C	E	BR	COMPARE INTEGER, BRANCH IF (A) + (X) EQ (Z)
3.2.1.178	B1	C	E	BR	COMPARE INTEGER, BRANCH IF (A) + (X) NE (Z)
3.2.1.179	B2	C	E	BR	COMPARE INTEGER, BRANCH IF (A) + (X) GE (Z)
3.2.1.180	B3	C	E	BR	COMPARE INTEGER, BRANCH IF (A) + (X) LT (Z)
3.2.1.181	B4	C	E	BR	COMPARE INTEGER, BRANCH IF (A) + (X) LE (Z)
3.2.1.182	B5	C	E	BR	COMPARE INTEGER, BRANCH IF (A) + (X) GT (Z)

If bit 0 of the G designator is cleared/set, registers A, X, C and Z are 64/32 bits respectively. Registers B and Y are always 64 bits.

G bits 1 and 2 must be set to zero.

These instructions are executed in the following 5 steps:

1. Form the sum of the 48-bit (24-bit if G bit 0 = 1) integers from the rightmost portion of registers A and X, ignoring overflows. If designators A and/or X equal zero, machine zero will be supplied.

2. Read register Z. If the Z designator is equal to zero, 48 zeros (24 zeros if G bit 0 = 1) are read as right-most bits.

3. Store the following in register C:

   - The sum from step 1 is stored into the rightmost 48 bits (24 bits if G bit 0 = 1) of register C.

   - The leftmost 16 bits (8 bits if G bit 0 = 1) of register A are copied into the leftmost portion of register C.

(continued)

3.2.1.182 (Cont'd)

4. Compare the sum formed in step 1 with register Z as follows:

   - o G bit 3 = 0 The integers compared are the 48-bit (24 bits if G bit 0 = 1) result of step 1 and the rightmost 48 bits (24 bits if G bit 0 = 1) read from register Z in step 2.
   - o G bit 3 = 1 The integers compared are the 64 bits that are stored into register C in step 3 and 64 bits read from register Z in step 2.

     This compare is defined only for the B0 and B1 instructions (EQ and NE).

     When both G bit 0 and G bit 3 are 1 the instructions are undefined.
   - o G bit 4 = 0 The integers compared are interpreted as signed two's complement numbers.
   - o G bit 4 = 1 The integers compared are interpreted as unsigned numbers.

The following table indicates the ordering of numbers from largest to smallest as controlled by G bit 4.

	0	1
Largest	7F -------- FF	FF -------- FF
	7F -------- FE	FF -------- FE
	.	.
	.	.
	.	.
	00 -------- 01	80 -------- 01
	00 -------- 00	80 -------- 00
	FF -------- FF	7F -------- FF
	.	.
	.	.
v	.	.
Smallest	80 -------- 01	00 -------- 01
	80 -------- 00	00 -------- 00

(continued)

2.1.182 (Cont'd)

5. If the specified compare condition is met the instruction performs as follows:

   - o G bit 5 = 0 Branch to the address formed by adding the halfword item count from register Y left shifted 5 places to the base address from register B.
   - o G bit 5 = 1 Branch to the address formed by adding (G bit 6 = 0) or subtracting (G bit 6 = 1) the halfword item counts from the B and Y designators (16 bits), left shifted 5 places, to the program address of this instruction.

If the specified compare condition is not met, the instructions will continue execution at the next sequential instruction.

If any of the following conditions occur, the operation of these instructions is undefined.

- o G bit 0 = 1 and G bit 3 = 1
- o G bit 3 = 1 for B2, B3, B4 and B5
- o G bit 5 = 0 and G bit 6 = 1
- o G bit 7 = 1

```
B0  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) EQ (Z)
B1  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) NE (Z)
B2  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) GE (Z)
B3  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) LT (Z)
B4  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) LE (Z)
B5  C  E  NT  COMPARE INTEGER, SET CONDITION IF (A) + (X) GT (Z)
```

If bit 0 of the G designator is cleared/set, registers A, X, Y, C and Z are 64/32 bits respectively. Register B is not used and must be set to zero.

G bit 1 = 0 and G bit 2 = 1

(continued)

3.2.1.182 (Cont'd)

These instructions are executed in 5 steps of which the first four (compare) steps are identical to the first four steps described for B0 through B5 instructions with G bits 1 and 2 equal to zero (Compare Branch)

If the specified compare condition is, met the instruction performs as follows:

Store into register Y and 64-bit quantity (32-bit if G bit 0 = 1) 000---001 and continue execution at the next sequential instruction.

If the specified compare condition is not met, the instruction performs as follows:

Store into register Y and 64-bit quantity (32-bit if G bit 0 = 1) 000---000 and continue execution at the next sequential instruction.

4. Compare the sum formed in step 1 with register Z as follows:

- o G bit 3 = 0 The integers compared are the 48-bit (24 bits if G bit 0 = 1) result of step 1 and the rightmost 48 bits (24 bits if G bit 0 = 1) read from register Z in step 2.

- o G bit 3 = 1 The integers compared are the 64 bits that are stored into register C in step 3 and 64 bits read from register Z in step 2.

  This compare is defined only for the B0 and B1 instructions (EQ and NE).

  When both G bit 0 and G bit 3 are 1 the instructions are undefined.

- o G bit 4 = 0 The integers compared are interpreted as signed two's complement numbers.

(continued)

3.2.1.182 (Cont'd)

o G bit 4 = 1 The integers compared are interpreted as unsigned numbers.

The following table indicates the ordering of numbers from largest to smallest as controlled by G bit 4.

	0	1
Largest	7F -------- FF	FF -------- FF
	7F -------- FE	FF -------- FE
	.	.
	.	.
	.	.
	00 -------- 01	80 -------- 01
	00 -------- 00	80 -------- 00
	FF -------- FF	7F -------- FF
	.	.
	.	.
v	.	.
Smallest	80 -------- 01	00 -------- 01
	80 -------- 00	00 -------- 00

If any of the following conditions occur, the operation of these instructions is undefined:

o G bit 0 = 1 and G bit 3 = 1

o G bit 3 = 1 for B2, B3, B4 and B5

o G bit 5 = 1, G bit 6 = 1 or G bit 7 = 1

o The C designator is equal to the Z designator

```
B0  C  E  BR  COMPARE F.P., BRANCH IF (A) EQ (X)
B1  C  E  BR  COMPARE F.P., BRANCH IF (A) NE (X)
B2  C  E  BR  COMPARE F.P., BRANCH IF (A) GE (X)
B3  C  E  BR  COMPARE F.P., BRANCH IF (A) LT (X)
B4  C  E  BR  COMPARE F.P., BRANCH IF (A) LE (X)
B5  C  E  BR  COMPARE F.P., BRANCH IF (A) GT (X)
```

If bit 0 of the G designaor is cleared/set, registers A and X are 64/32 bits respectively. Registers B and Y are always 64 bits. Registers C and Z are not used and must be set to zero.

G bit 1 = 1 and G bit 2 = 0

(continued)

3.2.1.182 (Cont'd)

These instructions compare the two floating point operands from registers A and X according to the floating point compare rules in Section 3.1.4.5.

If the specified compare condition is met, the instructions perform as follows:

- o G bit 5 = 0 Branch to the address formed by adding the halfword item count from register Y, left shifted 5 places, to the base address from register B.
- o G bit 5 = 1 Branch to the address formed by adding (G bit 6 = 0) or subtracting (G bit 6 = 1) the halfword item counts from the B and Y designators 16 bits), left shifted 5 places, to the program address of this instruction.

If the specified compare condition is not met, the instructions will continue execution at the next sequential instruction.

If any of the following conditions occur, the operation of these instructions is undefined:

- o G bit 3 = 1, G bit 4 = 1 or G bit 7 = 1
- o Designaor Z and/or C not equal to zero
- o G bit 5 = 0 and G bit 6 = 1

Data Flag: bit 46.

```
B0  C  E  NT  COMPARE F.P, SET CONDITION IF (A) EQ (X)
B1  C  E  NT  COMPARE F.P, SET CONDITION IF (A) NE (X)
B2  C  E  NT  COMPARE F.P, SET CONDITION IF (A) GE (X)
B3  C  E  NT  COMPARE F.P, SET CONDITION IF (A) LT (X)
B4  C  E  NT  COMPARE F.P, SET CONDITION IF (A) LE (X)
B5  C  E  NT  COMPARE F.P, SET CONDITION IF (A) GT (X)
```

If bit 0 of the G designator is cleared/set, registers A, X, and Y ae 64/32 bits respectively. Registers B, C and Z are not used and must be set to zero.

(continued)

3.2.1.182 (Cont'd)

G bit 1 = 1 and G bit 2 = 1

These instructions compare the two floating point operands from registers A and X according to the floating point compare rules in Section 3.1.4.5.

If the specified compare condition is met the instruction performs as follows:

Store into register Y and 64-bit quantity (32-bit if G bit 0 = 1) 000---001 and continue execution at the next sequential instruction.

If the specified compare condition is not met, the instruction performs as follows:

Store into register Y the 64-bit quantity (32-bit if G bit 0 = 1) 000---000 and continue execution at the next sequential instruction.

If any of the following conditions occur, the operation of these instructions is undefined:

- o Any one of G bits 3 through 7 is set
- o Designators B, Z and/or C are not equal to zero

Data Flag: bit 46.

3.2.1.183 B6 5 NA BR BRANCH TO IMMEDIATE ADDRESS;(R)+I(48 BITS)

The right-most 48 bits of register R contain an item count of half-words. The right-most 48 bits of the instruction word contain an immediate operand which is used as a base address. An unconditional branch is taken to the branch address formed by adding the item count to the base address (the item count is shifted left 5 places before the addition and overflow, if any, is ignored).

A direct branch is taken to the base address from the instruction word if the R designator is zero or if the right-most 43 bits of register R are zeros.

3.2.1.184 B7 1 E VM TRANSMIT LIST--->INDEXED C

This instruction scatters groups of elements from vector B into vector C. The locations of the element groups in vector C are specified by the item counts contained in the right-most 48 bits of each element of vector A. The first group of elements from vector B is transmitted to vector C beginning at the address formed by adding the first item count from vector A to the base address in register C. The second group of elements from vector B is then transmitted to vector X beginning at the address formed by adding the second item count from vector A to the base addressin register C. This continues until vector A is exhausted.

The elements of vector A are always 64-bit elements, while the elements of vectors B and C are 64-bit or 32-bit as a function of G-bit 0. Before the addition of the item count from A to the base address in register C, the item count is left-shifted 5 places for 32-bit operands and 6 places for 64 bit operands.

When G bit 5=1 vector A is replaced by a fixed increment specified by the rightmost 48 bits of register A. The addressing of vector C is then; C,CA,C+2A....,C+(N-1)A where N is the field length specified by the leftmost 16 bits of register A and still determines the total number of groups. The fixed increment A is shifted left 6 (G bit 0=0) or 5 (G bit 0=1) places before it is added to C.

The Y and Z designators are undefined and must be set to zero, thus there can be no offset for the B field nor control vector fo the C field. There are no field lengths for vectors B and C. The left-most 16 bits of register C are ignored except when used, as described below, to specify the number of elements in each group to be transmitted. Note that all groups contain the same number of elements.

(continued)

3.2.1.184 (Cont'd)

G-BIT	STATE	INTERPRETATION
0	0	Elements of vectors B and C are 64-bit.
	1	Elements of vectors B and C are 32-bit.
1-3		Undefined and must be set to zero.
*4	0	No broadcast of vector B.
	1	Broadcast vector B; permitted only with G-bit 6 set to zero.
5	0	Use Vector A
	1	Use fixed increment A (x designator must = 0)
**6	0	The number of elements in each group to be transmitted is fixed at one. Thus a single element from vector B is transmitted to vector C for each element of vector A. For this case, the left-most 16 bits of register C are ignored.
	1	The number of elements in each group to be transmitted is specified by the left-most 16 bits of register C. If the left-most 16 bits of register C are zero, this instruction is a no-op. (No Broadcast B if G-bit 6=1.)
**7	0	Vector C resides in central memory.
	1	All elements of output vector C must reside within the range of absolute or virtual addresses 0 through 3FC0. Reference to the register file as central memory in this case is therefore allowed. This instruction and the BA instruction are the only instructions which permit this type of reference to occur. If all addresses for vector C are not contained in the register file, this instruction is undefined.

* If both G-bit 4 and 6 are set, this instruction is undefined.

** If both G-bit 6 and 7 are set, this instruction is undefined.

3.2.1.185 B8 1 E VM TRANSMIT REVERSE;A--->C

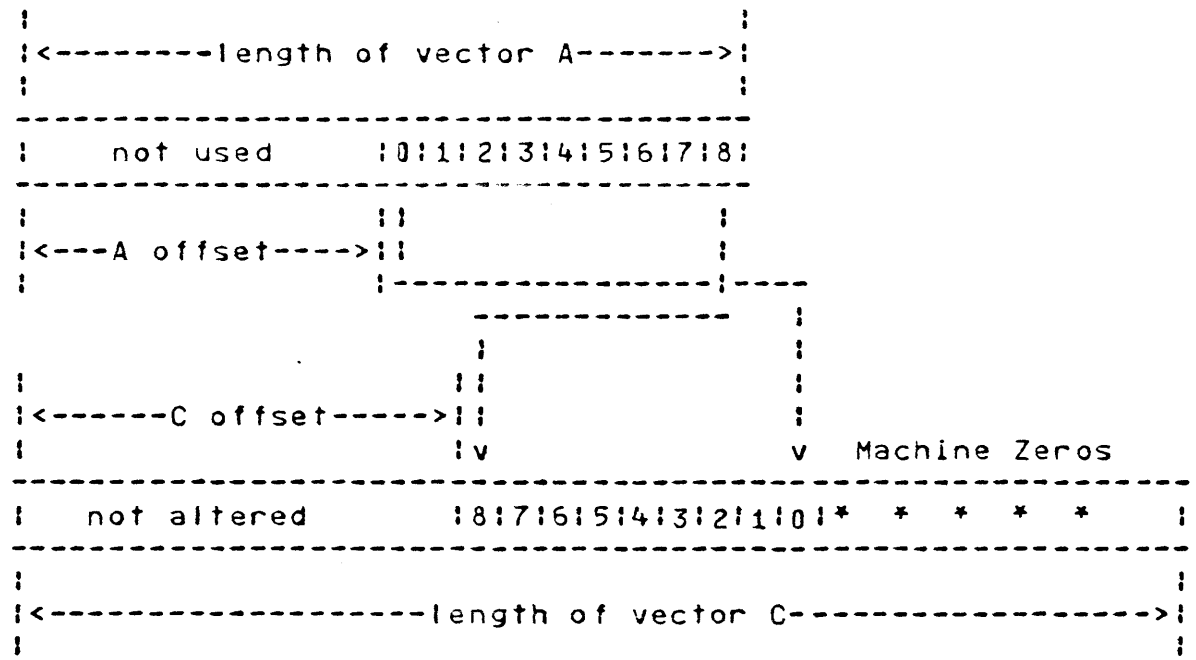
Transmit, in reverse order, vector A to vector C. The last element of vector A is the first element of vector C, the next to 1st element of vector A is the second element of vector C, etc.

Any overlap causes this instruction to be undefined.

The Y and B designators and bits 3-7 of the G designator are undefined and must be set to zeros.

This instruction terminates when vector C is exhausted.

Transmit Reverse Example



* Vector A is exhausted before vector C, so machine zeros are transferred to fill out the remainder of vector C.

For the above example, assume the Z designator was zero (no control vector used).

3.2.1.186 B9 ILLEGAL

3.2.1.187 BA 1 E VM TRANSMIT INDEXED LIST--->C

This instruction gathers groups of elements from vector B into vector C. The locations of the element groups in vector B are specified by the item counts contained in the right-most 48 bits of each element of vector A. The first group of elements transmitted to vector C come from vector B beginning at the address formed by adding the first item count from vector A to the base address in register B. The second group of elements transmitted to vector C come from vector B beginning at the address formed by adding the second item count from vector A to the base address in register B. The groups of elements are stored in vector C in consecutive order. This continues until vector A is exhausted.

The elements of vector A are always 64-bit elements, while the elements of vectors B and C are 64-bit or 32-bit as a function of G-bit 0. Before the addition of the item count from A to the base address in register B, the item count is left-shifted 5 places for 32-bit operands and 6 places for 64-bit operands.

When G bit 5=1 vector A is replaced by a fixed increment specified by the rightmost 48 bits of register A. The addressing of vector B is then; $B, B+A, B+2A, \ldots, B+(N-1)A$ where N is the field length specified by the leftmost 16 bits of register A and still determines the total number of groups. The fixed increment A is shifted left 6 (G bit 0=0) or 5 (G bit 0=1) places before it is added to B.

The Y and Z designators are undefined and must be set to zero, thus there can be no offset for the B field nor control vector for the C field. There are no field lengths for vectors B and C. The left-most 16 bits of register B are ignored except when used, as described below, to specify the number of elements in each group to be transmitted. Note that all groups contain the same number of elements.

Broadcasting is not used by this instruction.

(continued)

3.2.1.187 (Cont'd)

G-BIT	STATE	INTERPRETATION
0	0	Elements of vector B and C are 64-bit.
	1	Elements of vector B and C are 32-bit.
1-4		Undefined and must be set to zero.
5	0	Use vector A
	1	Use fixed increment A (x designator must = 0)
*6	0	The number of elements in each group to be transmitted is fixed at one. Thus a single element from vector B is transmitted to vector C for each element of vector A. For this case, the left-most 16 bits of register B are ignored
	1	The number of elements in each group to be transmitted is specified by the left-most 16 bits of register B. If the left-most 16 bits of register B are zero, this instruction is a no-op.
*7	0	Vector B resides in central memory.
	1	All elements of input vector B must reside within the range of absolute or virtual bit addresses 0 through 3FC0. Reference to the register file as central memory is therefore allowed. This instruction and the B7 are the only instructions which permit this type of reference to occur. If all addresses for vector B are not contained in the register file, this instruction is undefined. All references to Register Zero are to the Trace Register.

* If both G-bit 6 and 7 are set, this instruction is undefined.

3.2.1.188 B3 2 E NT MASK; A, B---->C PER Z

Elements of vector A and elements of vector B are merged to form result vector C as directed by order vector Z. When a binary one is encountered in order vector Z, the next element of vector A is inserted into result vector C and an element of vector B is skipped. When a binary zero is encountered in order vector Z, the next element of vector B is inserted into result vector C and an element of vector A is skipped. The resulting length of vector C is transmitted to the length specification portion of register C.

If bit 0 of the G designator is cleared/set, the operand size is 64/32, respectively. The X and Y designators and bits 1, 2, 5, 6, and 7 of the G designator are undefined and must be set to zero. Bits 3 and 4 of G are used to broadcast the constants (A) and (B) respectively.

This instruction terminates when order vector Z is exhausted. No lengths are recognized on vectors A and B.

3.2.1.189 BC 2 E NT COMPRESS; A---->C PER Z

Vector A is compressed by forming sparse data vector C which is composed of the elements of vector A associated with binary ones in sparse order vector Z, i.e., those elements of vector A in positions of binary ones (G bit 1 equal 0) in sparse order vector Z are selected and inserted, in order, into sparse data vector C. If G bit 1 is set, the elements of vector A in positions which correspond to the positions of binary zeros in sparse order vector Z are inserted in sparse data vector C.

The resulting length of sparse data vector C is transferred to the length specification portion of register C. If bit 0 of the G designator is cleared/set, the operand size is 64/32 bits, respectively. The X, Y and B designators and bits 2-7 of the G designator are undefined and must be set to zero.

(continued)

3.2.1.189 (Cont'd)

This instruction terminates when sparse order vector Z is exhausted. The length specification portion of registers A and C is ignored.

3.2.1.190 BD 2 E NT MERGE; A, B---->C PER Z

Elements of vector A and elements of vector B are merged to form result vector C as directed by order vector Z. When a binary one is encountered in order vector Z, the next element of vector A is inserted into result vector C. When a binary zero is encontered in order vector Z, the next element of vector B is inserted into result vector C. Note that no elements of A or B are skipped if G bit 7 is a zero. If G bit 7 is a one, the corresponding operand of B is skipped for each A operand stored, but A is not skipped on B stored. The resulting length of vector C is transmitted to the length specification portion of register C.

If bit 0 of the G designator is cleared/set, the operand size is 64/32, respectively. The X and Y designators and bits 1, 2, 5 and 6 of the G designator are undefined and must be set to zero. Bits 3 and 4 of G are used to broadcast the constants (A) and (B), respectively.

If G-bit 7 is a zero, the operation is called merge. If G-bit 7 is a one, the operation is called decompress.

If G-bit 3 or 4 is a one, the operation is called expand.

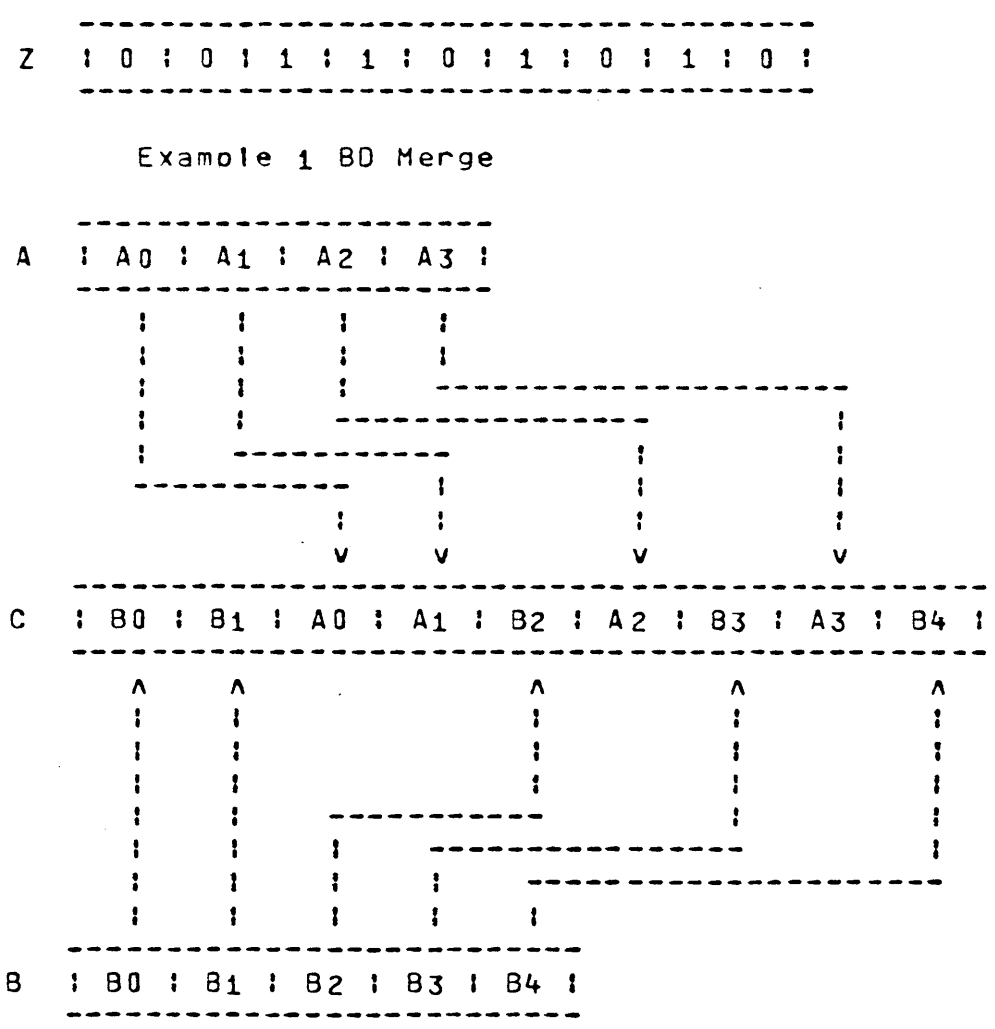
This instruction terminates when order vector Z is exhausted. No lengths are recognized on vectors A and B.

(continued)

3.2.1.190 (cont'd)

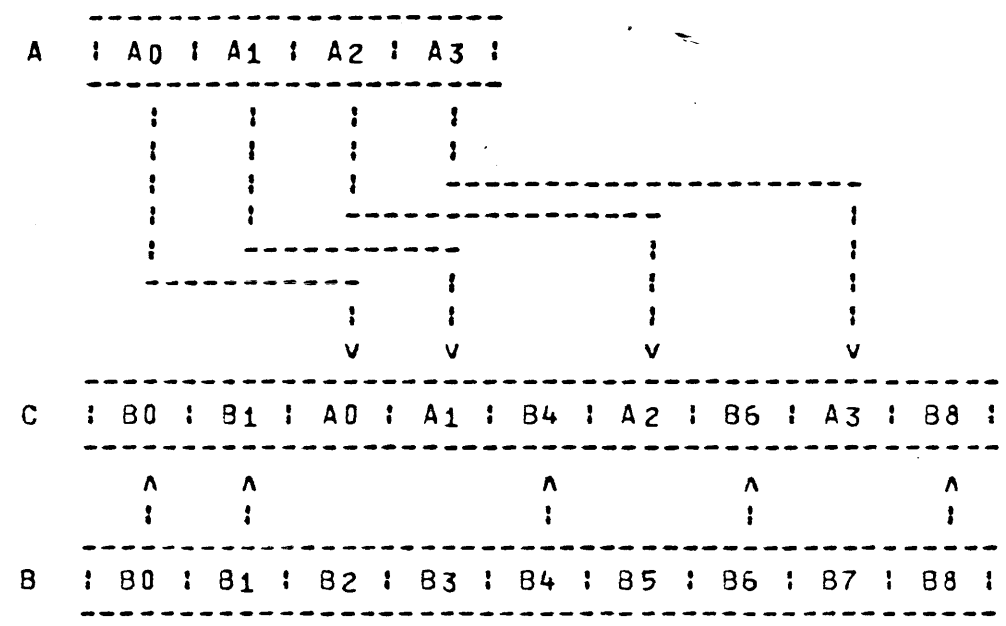
BD Merge Instruction Examples

The Z-bit string is used for all three examples.
G-bits not indicated are zeroes.

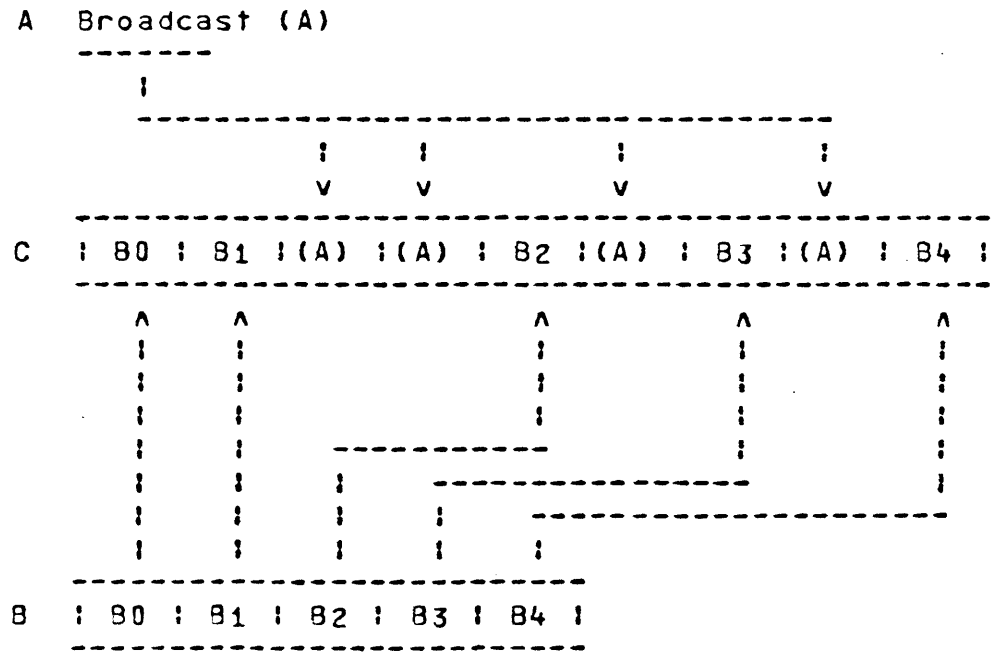


Examole 1 BD Merge

Example 2 BD Decompress G-bit 7=1



Example 3 BD Expand G-bit 3=1



3.2.1.191 BE 5 64 IN ENTER (R) WITH I(48 BITS)

Clear register r and transfer the right-most 48 bits of this instruction to the right-most 48 bits of register R.

3.2.1.192 BF 5 64 IN INCREASE (R) BY I(48 BITS)

Replace the right-most 48 bits of register R by the sum of those bits and the right-most 48 bits of this instruction word. Arithmetic overflow is ignored.

3.2.1.193 C0 1 E VM SELECT EQ; A EQ B, ITEM COUNT TO (C)
3.2.1.194 C1 1 E VM SELECT NE; A NE B, ITEM COUNT TO (C)
3.2.1.195 C2 1 E VM SELECT GE; A GE B, ITEM COUNT TO (C)
3.2.1.196 C3 1 E VM SELECT LT; A LT B, ITEM COUNT TO (C)

Each element of vector A is compared with its associated element of vector B. This operation proceeds until the compare condition (A = , $\neq$, $\geq$, < B) is met or until the shorter of the two vectors is exhausted. If broadcast is selected for field A or field B (but not both) the instruction will terminate when the non-broadcast field terminates.

If the compare condition is met, the item count is equal to the number of pairs of elements encountered up to (but not including the pair meeting the condition). If the compare condition is not met, the item count is equal to the length of the shorter vector (where the shorter vector's length is determined after the offset adjustment). The item count is stored into the right-most 48 bits of a cleared register C.

The control vector, if used, determines which pairs of elements are compared. The item count, as described above, includes all pairs of elements encountered, not only those which were compared. If a control vector is used and either vector A or B is exhausted before a permissive control vector element is encountered, no compares are made. In this case, the item count stored is the length of the shorter vector minus its offset.

(continued)

3.2.1.196 (Cont'd)

Each element of vector B is subtracted from the corresponding element of vector A. The operational decision is made on the result of this subtract according to the "floating point compare rules" in 3.1.4.5

Bits 2, and 5-7 of the G designator are undefined and must be set to zero.

If the C designator is zero, the results of this instruction are undefined.

Data flags: bits 37 and 46.

3.2.1.197	C4	1	E	NT	COMPARE EQ; A EQ B ORDER VECTOR ----> Z
3.2.1.198	C5	1	E	NT	COMPARE NE; A NE B ORDER VECTOR ----> Z
3.2.1.199	C6	1	E	NT	COMPARE GE; A GE B ORDER VECTOR ----> Z
3.2.1.200	C7	1	E	NT	COMPARE LT; A LT B ORDER VECTOR ----> Z

Successive elements of vector A are compared with successive elements of vector B. If the compare condition (A =, ≠, ≥, < B) specified by the instruction is met, the corresponding bit pf the result order vector A is set. If the compare condition is not met, the corresponding bit of Z is cleared. The instruction terminates when the Z field is filled.

The bits of the G designator are interpreted as follows:

Bit	State	Interpretation
0	0	operands are 64 bits long (words)
	1	operands are 32 bits long (half words)
3	0	normal vector A
	1	broadcast the constant in register A
4	0	normal vector B
	1	broadcast the constant in register B

The C designator and bits 1, 2, 5, 6 and 7 of the G designator are undefined and must be set to zero.

(continued)

3.2.1.200 (Cont'd)

Registers X and Y contain offsets for vectors A and B, respectively. When a constant is broadcast for either vector, that vector has no length and the offset is ignored.

The lengths and base addresses of vectors A, B and Z are contained in registers A, B, and Z, respectively. The lengths of vectors A and B are in words or half-words. The length of vector Z is in bits.

Each element of vector B is subtracted from the corresponding element of vector A. The operational decision is made on the result of this subtract according to the "floating point compare rules" in 3.1.4.5.

Data flag: bit 46

3.2.1.201	C8	1	E	NT	SEARCH EQ;	INDEX LIST ---->C
3.2.1.202	C9	1	E	NT	SEARCH NE;	INDEX LIST ---->C
3.2.1.203	CA	1	E	NT	SEARCH GE;	INDEX LIST ---->C
3.2.1.204	CB	1	E	NT	SEARCH LT;	INDEX LIST ---->C

For each element of vector A, search and compare against the successive elements of vector B. Terminate each search iteration when a Hit (A =, ≠, ≥, < B) is made or when vector B has been exhausted. After each iteration, clear the element in result vector C and transmit to it the index of the element in vector B which caused the search iteration to terminate. Regardless of whether 32 or 64-bit operands are used, the resulting index is a 64-bit word with the index in the right-most 48 bits. The left-most 16 bits are cleared to zero. For example, the sixth index in vector C appropriately shifted and added to the address of the first element of vector B will form the address of that element of vector B which caused the search iteration associated with the sixth element of A to terminate. The instruction terminates when vector A is exhausted.

G Designator Bits

Bit	State	Interpretation
0	0	A and B operands are 64 bits long.
	1	A and B operands are 32 bits long.

(continued)

3.2.1.204 (Cont'd)

Bit	State	Interpretation
1	0	Control vector operates (allows store into the C vector) on binary ones.
	1	Control vector operates on binary zeros.
2	0	Start each search iteration at the beginning of vector B.
	1	Start each search iteration at the location of the hit found in the previous search iteration.

A control vector (see Section 3.1.1.2.4) may be specified by the Z designator with each bit of the control vector associated with a single element of vector C (thus controlling the storage of an index into that specific element). No length nor offset is recognized for the control vector. This instruction performs as if a search iteration is performed for each element of vector A regardless of the control vector.

The end of vector B acts like a hit, thus the index stored for a search iteration which exhausts vector B will be equal to the length of vector B. Note that if G bit 2 = 1, all following search iterations will start and end at the end of vector B. If the length of vector B is initially zero, all indices stored will be zero.

For either the case of vector B being exhausted with G bit 2 = 1 or the length of vector B being initially zero, search iterations for each element of vector A will continue to be performed until vector A is exhausted. Thus, an indefinite element anywhere in vector A will always cause Data Flag Bit 46 to be set.

The X and Y designators and bits 3-7 of the G designator are undefined and must be set to zero. No lengths nor offsets are recognized on vectors C and Z.

Each element of vector B is subtracted from the corresponding element of vector A. The operational

(continued)

3.2.1.204 (Cont'd)

decision is made on the result of this subtract according to the "floating point compare rules"2 in 3.1.4.5.

Data flags: bit 46

3.2.1.205 CC 3 64 NT MASKED BINARY COMPARE; A EQ/NE (B) PER (C)

This instruction searches source field A (Reference Field) for a match with the contents of the register specified by the B designator. The contents of the register specified by the C designator serves as a word mask such that the instruction makes a word-by-word comparison only when there are ones in the corresponding bit positions of the mask. Bits of the reference field and the contents of B are considered to match wherever there is a zero bit in the mask word.

The Y and Z designators and G-Bits 0-5 are not used and must be zeroes. G-Bit 7 = 0 and 1 to search for equality and inequality, respectively. Registers B and C are 64-bit quantities.

The A index is incremented by one after each word searched not resulting in a match. However, if no match is found, the A index is increased by the length of the A field. When a match is found, the A index provides a means of locating the word of the reference field matching the contents of B.

Index increments for mask binary compare.

Field	Data Flag Bit 37	Index Increment
A	1	Full Increment (No Match)
A	0	Partial Increment (Match)

Data flag: bit 37

3.2.1.206 CD 5 32 IN HALF WORD ENTER (R) WITH I(24 BITS)

Clear register R and transfer the right-most 24 bits of this instruction to the right-most 24 bits of register R.

3.2.1.207 CE 5 32 IN HALF WORD INCREASE (R) BY I(24 BITS)

Replace the right-most 24 bits of register R by the sum of those bits and the right-most 24 bits of this instruction word. Arithmetic overflow is ignored.

3.2.1.208 CF 1 E NT ARITH. COMPRESS; A---->C PER B

Sparse data vector C and its associated sparse order vector Z are formed by performing a floating point compare operation between elements of vector A and elements of vector B. For elements of vector A whose value is greater than or equal to the associated element of vector B, the element of vector A becomes an element of sparse data vector C and the associated sparse order vector bit is made a one. For elements of vector A whose value is less than the associated element of vector B, no element is stored (or skipped) in sparse data vector C and the associated sparse order vector bit is cleared to zero. Note that the sign control bits of the G field may specify operations on the elements of vector A and/or B before the "floating point compare" is made; however, the element of A, if stored into C, will be the original element as read from vector A. Registers X and Y contain offsets for the A and B vectors respectively.

If bit 0 of the G designator is cleared/set, the operand size is 64/32 bits, respectively. If bit 4 of the G designator is set, register B contains a constant which is broadcast for vector B. In this case, the Y designator is ignored. Bits 5, 6 and 7 of the G designator specify sign control; see section 3.1.4.9 for details. Bits 1, 2, and 3 of the G designator are undefined and must be set to zero.

This instruction terminates when vector A is exhausted. Upon termination, the number of operations performed (the bit length of the generated sparse order vector) is stored into the length portion of register Z and the number of operands copied into sparse data vector C is stored into the length portion of register C. The Z and C register results are undefined if the Z and C designators are equal.

The B field is extended with machine zero when the B field length is exhausted.

(continued)

3.2.1.208 (Cont'd)

Each element of vector B is subtracted from the corresponding element of vector A. The operational decision is made on the result of this subtract according to the "floating point compare rules" in 3.1.4.5.

Data flag: bit 46

Arithmetic compress example (broadcast B)

F	G	X	A	Y	B	Z	C
CF 0	8:0	5:0	6:0	0:0	7:0	8:0	9:

Before execution

Register 05 = 0000000000000001
06 = 0007000000010000
07 = floating point constant B
08 = 0000000000020000
09 = 0000000000030000

Bit address 10000 - 10180

A 0	A 1	A 2	A 3	A 4	A 5	A 6

_ __/
v
64 bits

where :A 0: is not examined due to the offset of 1 from register 05

:A 1: > B
: -

:A 2: < B

:A 3: > B
: -

:A 4: < B

:A 5: > B
: -

(continued)

3.2.1.208 (Cont'd)

```
:A  : > B
: 61 -
```

After execution

Register 05, 06 and 07 are unchanged.
08 = 0006000000020000
09 = 0004000000030000

Bit address 20000 - 20005

```
-------------------------
: 1 : 0 : 1 : 0 : 1 : 1 :
-------------------------
                    \_ _/
                      v
                    1 bit
```

Bit address 30000 - 300C0

```
---------------------
: a  : a  : a  : a  :
:  1 :  3 :  5 :  6 :
---------------------
                \_ __/
                   v
                64 bits
```

3.2.1.209 D0 1 E VM AVERAGE (A(N)+B(N))/2---->C(N)

The Nth element of result vector C is the normalized sum of the Nth elements of vectors A and B divided by two. Dividing by two is accomplished by reducing the exponent of the sum by one.

Bits 5-7 of G designator are undefined and must be set to zero.

Data flags: Bits 43 and 46

3.2.1.210 D1 1 E VM ADJ. MEAN (A(N+1)+A(N))/2----> C(N)

The Nth element of result vector C is the normalized sum of the Nth and Nth + 1 elements of vector A divided by two. Dividing by two is accomplished by reducing the exponent of the sum by one.

The Y and B designators and bits 3-7 of the G designator are undefined and must be set to zero.

Data flags: bits 43 and 46

SUPER COMPUTER OPERATIONS

3.2.1.211 D2 ILLEGAL
3.2.1.212 D3 ILLEGAL

3.2.1.213 D4 1 E VM AVE. DIFF. (A(N)-B(N))/2---->C(N)

The Nth element of result vector C is the normalized difference of the Nth elements of vectors A and B divided by two. Dividing by two is accomplished by reducing the exponent of the difference by one.

Bits 5-7 of the G designator are undefined and must be set to zeros.

Data flag: bits 43 and 46

3.2.1.214 D5 1 E VM DELTA (A(N+1)-A(N))---->C(N)

The Nth element of result vector C is formed by subtracting the Nth element of vector A from the Nth + 1 element of vector A. Normalized arithmetic is used.

The Y and B designators and bits 3-7 of the G designator are undefined and must be set to zero.

3.2.1.215 D6 ILLEGAL
3.2.1.216 D7 ILLEGAL
3.2.1.217 D8 1 E NT MAX. OF A TO (C), ITEM COUNT TO (B)
3.2.1.218 D9 1 E NT MIN. OF A TO (C), ITEM COUNT TO (B)

Search and compare (using floating point compare rules) the successive elements of vector A for the maximum element and transmit it to register C. The number of elements in vector A before (but not including) the maximum element is the item count which is stored into the right-most 48 bits of a cleared register B. The instruction terminates when vector A is exhausted.

In the event of multiple maximum elements, data flag 54 will be set and the first of the multiple maximum elements examined will be the one recorded. In this case these elements, although equal, are not necessarily identical.

If an indefinite element is encountered and examined, register C is set to indefinite and data flag bit 46 is set. In this case, the contents of register B and data flag bit 54 are undefined.

(continued)

3.2.1.218 (Cont'd)

G Designator Bits

Bit	State	Interpretation
0	0	A operands and register C are 64 bit
	1	A operands and register C are 32 bit
1	0	Control vector operates (causes the element of A to be examined) on binary ones
	1	Control vector operates on binary zeros
5	0	
	1	Sign control (see section 3.1.4.9)

A control vector may be specified by the Z designator with each bit of the control vector associated with a single element of vector A (thus controlling the elements of vector A which are examined). No offset nor length is defined for the control vector. If the control vector is used and it has no permissive elements in it, no elements of vector A are examined and the contents of register C are undefined. In this case, the item count in register B is the length of vector A minus the A offset.

The length and base address of vector A are in register A. Register X contains the offset for vector A.

One of the Sign Control (section 3.1.4.9) operations is available by the use of G-bit 5. By setting this bit, the magnitude of the elements of vector A are compared. The unaltered element as read from vector A will be stored into register C.

The Y designator and bits 2, 3, 4, 6 and 7 of the G designator are undefined and must be set to zero.

The B and C Register results are undefined if the B and C designaors are equal.

The 09 (Minimum of A to C) instruction is identical to the preceding description with the word minimum substituted for maximum.

Data flags: bits 46 and 54.

3.2.1.219 DA 1 E VM SUM (A0+A1+A2...AN) TO C AND C+1

The double precision unnormalized floating point sum of all the elements of vector A is placed into the registers designated by C and C+1. The Upper Result and the Lower Result are stored into registers C and C+1, respectively. The instruction terminates when vector A is exhausted. Register C must be even. If register C is odd or zero, the instruction results are undefined.

Data flag bit 43 is determined only by the final result and will be set if the Lower Result is machine zero, regardless of the value of the upper result. If the Upper Result is indefinite, the Lower Result is undefined. Data flag bits 42 and 46 will be set normally as required on any one of the Add operations.

If a control vector is specified and contains no permissive elements, the result is machine zero and Data Flag 43 is set.

The Y and B designators and bits 2-7 of the G designator are undefined and must be set to zero. There is no length specification nor offset for control vector A.

Data flags: bits 42, 43 and 46

3.2.1.220 DB 1 E VM PRODUCT; (A0, A1, A2...AN) TO C

This instruction forms the Significant Product of the successive elements of vector A and stores it into register C. The number of significant bits in the partial product is adjusted after each multiplication.

Data flag bits 43 and 46 are determined only by the final result. Data flag bit 42 will be set if any multiply operation overflows.

The Y and B designators and bits 2-7 of the G designator are undefined and must be set to zero. There is no length specification for control vector Z. The instruction terminates when vector Z is exhausted.

(continued)

3.2.1.220 (Cont'd)

If the C designator is equal to zero, the results of this instruction are undefined.

If the control vector contains no permissive elements the result is a normalized one.

Data flags: bits 42, 43 and 46

3.2.1.221 DC 1 E VM DOT PRODUCT TO (C) AND (C+1)

Multiply vector A by vector B and form the sum of the products. Double precision, unnormalized arithmetic is performed. Bits 2-7 of the G designator are undefined and must be set to zero.

The Upper Result and the Lower Result are stored in the registers designated by C and C+1, respectively.

Data flag bits 43 and 46 are determined only by the final Upper and Lower Result. If the Upper Result is indefinite, the Lower Result is undefined. Data flag bit 43 will be set if the Lower Result is machine zero, regardless of the value of the upper result. Data flag bit 42 will be set if any multiply or addition operation overflows.

There is no length specification for control vector Z.

Register C must be even. If register C is odd or zero, the instruction results are undefined.

If the control vector contains no permissive elements the result is machine zero and Data flag 43 is set.

Data flags: bits 42, 43 and 46

3.2.1.222 DD ILLEGAL

3.2.1.223 DE ILLEGAL

3.2.1.224 DF 1 E VM INTERVAL;A PER B---->C

This instruction forms a result vector C whose initial element is the constant from register B and whose succeeding elements are greater than the preceding element of vector C by the constant contained in register B. Thus, the second element equals the first element of C plus the contents of B; the third element equals the second element plus the contents of B, etc. Arithmetic is unnormalized.

If a control vector is used, the "last element" is the last element formed even though it was not stored into the destination field.

If a non-permissive bit in the control vector is encountered, the addition operation is performed and the result retained but not stored in the result vector. If the result of this operation is indefinite, the appropriate data flag will not be set until a permissive bit is encountered in the control vector thus allowing a result to be stored in the result vector. Overflow will be set on the next permitted store even if the iterative step which overflowed was not stored.

If the A designator is zero, this is treated as a broadcast register and 8000----0 is read from the register zero. The X and Y designators and bits 3 through 7 of the G designator are undefined and must be set to zero.

Data flags: bits 42, 43 and 46

3.2.1.225 E0 ILLEGAL
3.2.1.226 E1 ILLEGAL
3.2.1.227 E2 ILLEGAL
3.2.1.228 E3 ILLEGAL
3.2.1.229 E4 ILLEGAL
3.2.1.230 E5 ILLEGAL
3.2.1.231 E6 ILLEGAL
3.2.1.232 E7 ILLEGAL
3.2.1.233 E8 ILLEGAL
3.2.1.234 E9 ILLEGAL
3.2.1.235 EA ILLEGAL
3.2.1.236 EB ILLEGAL
3.2.1.237 EC ILLEGAL
3.2.1.238 ED ILLEGAL
3.2.1.239 EE ILLEGAL
3.2.1.240 EE ILLEGAL
3.2.1.241 F0 3 1 LS LOGICAL EXCLUSIVE OR A, B---->C

------- SUPER COMPUTER OPERATIONS --------

3.2.1.242	F1	3	1	LS	LOGICAL AND	A, B---->C
3.2.1.243	F2	3	1	LS	LOGICAL INCLUSIVE OR	A, B---->C
3.2.1.244	F3	3	1	LS	LOGICAL STROKE	A, B---->C
3.2.1.245	F4	3	1	LS	LOGICAL PIERCE	A, B---->C
3.2.1.246	F5	3	1	LS	LOGICAL IMPLICATION	A, B---->C
3.2.1.247	F6	3	1	LS	LOGICAL INHIBIT	A, B---->C
3.2.1.248	F7	3	1	LS	LOGICAL EQUIVALENCE	A, B---->C

The above instrucions perform the indicated bit by bit logical functions on binary fields A and B and store the result into field C.

TRUTH TABLE

		EXCL. OR	AND	OR	STROKE	PIERCE	IMPLI.	INHIBIT	EQUIV.
A	B	A-B	A.B	A+B	$\overline{(A.B)}$	$\overline{(A+B)}$	$A+\overline{B}$	$A.\overline{B}$	$A-\overline{B}$
0	0	0	0	0	1	1	1	0	1
0	1	1	0	1	1	0	0	0	0
1	0	1	0	1	1	0	1	1	0
1	1	0	1	1	0	0	1	0	1

Binary field A, B and C are strings of bits. The operation proceeds from left to right and terminates when the C field is exhausted. Item counts are bit counts.

Fields A and/or B are extended automatically with binary zeros if they are shorter than field C.

The G designator is undefined and must be set to zeros.

Data flags: Result field all zeros bit 53, result field mixed bit 54, and result field all ones bit 55.

3.2.1.249 F8 3 8 ST MOVE BYTES LEFT; A---->C

This instruction moves source field A to result field C. The bytes in the field are considered from left to right. Thus, the most significant byte of the source field is moved to the most significant byte position of the result field.

(continued)

3.2.1.249 (Cont'd)

The Y and G designators are not used and must be zeroes.

If the origin field is shorter than the destination field, the destination field is filled in with the repeated byte found in the B designator of the instruction.

If the origin field is longer than the destination field, the operation is truncated when the destination field is exhausted.

3.2.1.250	F9	ILLEGAL
3.2.1.251	FA	ILLEGAL
3.2.1.252	FB	ILLEGAL
3.2.1.253	FC	ILLEGAL
3.2.1.254	FD	ILLEGAL
3.2.1.255	FE	ILLEGAL
3.2.1.256	FF	ILLEGAL

4.0 TEST REQUIREMENTS (not applicable)

5.0 PREPARATION FOR DELIVERY (not applicable)

6.0 NOTES

6.1 ASCII/EBCDIC Reference Charts

The following table defines the control characters used in the ASCII Reference Chart.

NUL Null	DLE Data Link Escape (CC)
SOH Start of Heading (CC)	DC1 Device Control 1
STX Start of Text (CC)	DC2 Device Control 2
ETX End of Text (CC)	DC3 Device Control 3
EOT End of Transmission (CC)	DC4 Device Control 4 (Stop)
ENQ Enquiry (CC)	NAK Negative Acknowledge (CC)
ACK Acknowledge (CC)	SYN Synchronous Idle (CC)
BEL Bell (audible or attention signal)	ETB End of Transmission Block (CC)
BS Backspace (FE)	CAN Cancel
HT Horizontal Tabulation (punched card skip (FE)	EM End of Medium
	SUB Substitute
LF Line Feed (FE)	
	ESC Escape
VT Vertical Tabulation (FE)	
	FS File Separator (IS)
FF Form Feed (FE)	
	GS Group Separator (IS)
CR Carriage Return (FE)	
	RS Record Separator (IS)
SO Shift Out	
	US Unit Separator (IS)
SI Shift In	
	DEL Delete [1]

NOTE: (CC) Communication Control
(FE) Format Effector
(IS) Information Separator

[1] In the strict sense, DEL is not a control character.

ASCII REFERENCE CHART

b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1	ROW \ COL. →	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL	DLE	SP	0	@	P	\	p								
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q								
0 0 1 0	2	STX	DC2	"	2	B	R	b	r								
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s								
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t								
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u								
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v								
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w								
1 0 0 0	8	BS	CAN	(	8	H	X	h	x								
1 0 0 1	9	HT	EM	)	9	I	Y	i	y								
1 0 1 0	10 (A)	LF	SUB	*	:	J	Z	j	z								
1 0 1 1	11 (B)	VT	ESC	+	;	K	[	k	{								
1 1 0 0	12 (C)	FF	FS	,	<	L	\	l	¦								
1 1 0 1	13 (D)	CR	GS	-	=	M	]	m	}								
1 1 1 0	14 (E)	SO	RS	.	>	N	^	n	~					'			
1 1 1 1	15 (F)	SI	US	/	?	O	_	o	DEL								EO

EBCDIC REFERENCE CHART

b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1	ROW \ COL. →	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0 0 0 0	0	NUL	DLE			SP	&	–						{	}	\	0
0 0 0 1	1	SOH	DC1					/		a	j	~		A	J		1
0 0 1 0	2	STX	DC2		SYN					b	k	s		B	K	S	2
0 0 1 1	3	ETX	DC3							c	l	t		C	L	T	3
0 1 0 0	4									d	m	u		D	M	U	4
0 1 0 1	5	HT		LF						e	n	v		E	N	V	5
0 1 1 0	6		BS	ETB						f	o	w		F	O	W	6
0 1 1 1	7	DEL		ESC	EOT					g	p	x		G	P	X	7
1 0 0 0	8		CAN							h	q	y		H	Q	Y	8
1 0 0 1	9		EM						\	i	r	z		I	R	Z	9
1 0 1 0	10 (A)					[	]	¦	:								
1 0 1 1	11 (B)	VT				.	$	,	#								
1 1 0 0	12 (C)	FF	FS		DC4	<	*	%	@								
1 1 0 1	13 (D)	CR	GS	ENQ	NAK	(	)	_	'								
1 1 1 0	14 (E)	SO	RS	ACK		+	;	>	=								
1 1 1 1	15 (F)	SI	US	BEL	SUB	!	^	?	"								EO

AMERICAN NATIONAL STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII) WITH PUNCHED CARD CODES AND EBCDIC TRANSLATION

b4 b3 b2 b1	ROW \ COL	0 (b8 b7 b6 b5 = 0 0 0 0)	1 (0 0 0 1)	2 (0 0 1 0)	3 (0 0 1 1)	4 (0 1 0 0)	5 (0 1 0 1)	6 (0 1 1 0)	7 (0 1 1 1)	8 (1 0 0 0)	9 (1 0 0 1)	10 (A) (1 0 1 0)	11 (B) (1 0 1 1)	12 (C) (1 1 0 0)	13 (D) (1 1 0 1)	14 (E) (1 1 1 0)	15 (F) (1 1 1 1)
0 0 0 0	0	NUL 12-0-9-8-1 00	DLE 12-11-9-8-1 10	SP no punch 40	0 0 F0	@ 8-4 7C	P 11-7 D7	` 8-1 79	p 12-11-7 97	11-0-9-8-1 20	12-11-0-9-8-1 30	12-0-9-1 41	12-11-9-8 58	12-11-0-9-8 76	12-11-8-7 9F	12-11-0-8 BB	12-11-9-8-4 DC
0 0 0 1	1	SOH 12-9-1 SOH 01	DC1 11-9-1 DC1 11	! 12-8-7 4F	1 1 F1	A 12-1 A C1	Q 11-8 Q D8	a 12-0-1 a 81	q 12-11-8 q 98	0-9-1 SOS 21	9-1 31	12-0-9-2 42	11-8-1 59	12-11-0-9-7 77	11-0-8-1 A0	12-11-0-9 B9	12-11-9-8-5 DD
0 0 1 0	2	STX 12-9-2 STX 02	DC2 11-9-2 DC2 12	" 8-7 7F	2 2 F2	B 12-2 B C2	R 11-9 R D9	b 12-0-2 b 82	r 12-11-9 r 99	0-9-2 FS 22	11-9-8-2 CC 1A	12-0-9-3 43	11-0-9-2 62	12-11-0-9-8 78	11-0-8-2 AA	12-11-0-8-2 BA	12-11-9-8-6 DE
0 0 1 1	3	ETX 12-9-3 ETX 03	DC3 11-9-3 TM 13	# 8-3 = 7B	3 3 F3	C 12-3 C C3	S 0-2 S E2	c 12-0-3 c 83	s 11-0-2 s A2	0-9-3 23	9-3 33	12-0-9-4 44	11-0-9-3 63	12-0-8-1 80	11-0-8-3 AB	12-11-0-8-3 BB	12-11-9-8-7 DF
0 1 0 0	4	EOT 9-7 EOT 37	DC4 9-8-4 DC4 3C	$ 11-8-3 S 5B	4 4 F4	D 12-4 D C4	T 0-3 T E3	d 12-0-4 d 84	t 11-0-3 t A3	0-9-4 BYP 24	9-4 PN 34	12-0-9-5 45	11-0-9-4 64	12-0-8-2 8A	11-0-8-4 AC	12-11-0-8-4 BC	11-0-9-8-2 EA
0 1 0 1	5	ENQ 0-9-8-5 ENQ 2D	NAK 9-8-5 NAK 3D	% 0-8-4 6C	5 5 F5	E 12-5 E C5	U 0-4 U E4	e 12-0-5 e 85	u 11-0-4 u A4	11-9-5 NL 15	9-5 RS 35	12-0-9-6 46	11-0-9-5 65	12-0-8-3 8B	11-0-8-5 AD	12-11-0-8-5 BD	11-0-9-8-3 EB
0 1 1 0	6	ACK 0-9-8-6 ACK 2E	SYN 9-2 SYN 32	& 12 & 50	6 6 F6	F 12-6 F C6	V 0-5 V E5	f 12-0-6 f 86	v 11-0-5 v A5	12-9-6 LC 06	9-6 UC 36	12-0-9-7 47	11-0-9-6 66	12-0-8-4 8C	11-0-8-6 AE	12-11-0-8-6 BE	11-0-9-8-4 EC
0 1 1 1	7	BEL 0-9-8-7 BEL 2F	ETB 0-9-6 ETB 26	' 8-5 7D	7 7 F7	G 12-7 G C7	W 0-6 W E6	g 12-0-7 g 87	w 11-0-6 w A6	11-9-7 IL 17	12-9-8 GE 08	12-0-9-8 48	11-0-9-7 67	12-0-8-5 8D	11-0-8-7 AF	12-11-0-8-7 BF	11-0-9-8-5 ED
1 0 0 0	8	BS 11-9-6 BS 16	CAN 11-9-8 CAN 18	(12-8-5 4D	8 8 F8	H 12-8 H C8	X 0-7 X E7	h 12-0-8 h 88	x 11-0-7 x A7	0-9-8 28	9-8 38	12-8-1 49	11-0-9-8 68	12-0-8-6 8E	12-11-0-8-1 B0	12-0-9-8-2 CA	11-0-9-8-6 EE
1 0 0 1	9	HT 12-9-5 HT 05	EM 11-9-8-1 EM 19	) 11-8-5 5D	9 9 F9	I 12-9 I C9	Y 0-8 Y E8	i 12-0-9 i 89	y 11-0-8 y A8	0-9-8-1 29	9-8-1 39	12-11-9-1 51	0-8-1 69	12-0-8-7 B1	12-11-0-1 B1	12-0-9-8-3 CB	11-0-9-8-7 EF
1 0 1 0	10 (A)	LF 0-9-5 LF 25	SUB 9-8-7 SUB 3F	* 11-8-4- 5C	8-2 7A	J 11-1 J D1	Z 0-9 Z E9	j 12-11-1 j 91	z 11-0-9 z A9	0-9-8-2 SM 2A	9-8-2 3A	12-11-9-2 52	12-11-0 70	12-11-8-1 90	12-11-0-2 B2	12-0-9-8-4 CC	12-11-0-9-8-2 (LVM) FA
1 0 1 1	11 (B)	VT 12-9-8-3 VT 0B	ESC 0-9-7 ESC 27	+ 12-8-6 4E	11-8-6 5E	K 11-2 K D2	12-8-2 ¢ 4A	k 12-11-2 k 92	[12-0 C0	0-9-8-3 CU2 2B	9-8-3 CU3 3B	12-11-9-3 53	12-11-0-9-1 71	12-11-8-2 9A	12-11-0-3 B3	12-0-9-8-5 CD	12-11-0-9-8-3 FB
1 1 0 0	12 (C)	FF 12-9-8-4 FF 0C	FS 11-9-8-4 IFS 1C	, 0-8-3 68	< 12-8-4 < 4C	L 11-3 L D3	\ 0-8-2 E0	l 12-11-3 l 93	\ 12-11 6A	0-9-8-4 2C	12-9-4 PF 04	12-11-9-4 54	12-11-0-9-2 72	12-11-8-3 9B	12-11-0-4 B4	12-0-9-8-6 CE	12-11-0-9-8-4 FC
1 1 0 1	13 (D)	CR 12-9-8-5 CR 0D	GS 11-9-8-5 IGS 1D	- 11 60	8-6 7E	M 11-4 M D4	] 11-8-2] 5A	m 12-11-4 m 94	] 11-0 DD	12-9-8-1 IRF 09	11-9-4 RES 14	12-11-9-5 55	12-11-0-9-3 73	12-11-8-4 9C	12-11-0-5 B5	12-0-9-8-7 CF	12-11-0-9-8-5 FD
1 1 1 0	14 (E)	SO 12-9-8-6 SO 0E	RS 11-9-8-6 IRS 1E	. 12-8-3 4B	> 0-8-6 > 6E	N 11-5 N D5	^ 11-8-7 ^ 5F	n 12-11-5 n 95	^ 11-0-1 A1	12-9-8-2 SMM 0A	9-8-6 3E	12-11-9-6 56	12-11-0-9-4 74	12-11-8-5 9D	12-11-0-6 B6	12-11-9-8-2 DA	12-11-0-9-8-6 FE
1 1 1 1	15 (F)	SI 12-9-8-7 SI 0F	US 11-9-8-7 IUS 1F	/ 0-1 61	? 0-8-7 ? 6F	O 11-6 O D6	_ 0-8-5 6D	o 12-11-6 o 96	DEL 12-9-7 DEL 07	11-9-8-3 CU1 1B	11-0-9-1 E1	12-11-9-7 57	12-11-0-9-5 75	12-11-8-6 9E	12-11-0-7 B7	12-11-9-8-3 DB	EO 12-11-0-9-8-7 FF

LEGEND

EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE (EBCDIC) WITH PUNCHED CARD CODES AND ASCII TRANSLATION

BITS 4 5 6 7	1ST HEX 2ND	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
	BITS 0 1 2 3	0	1	2	3	4	5	6	7	8	9	A (10)	B (11)	C (12)	D (13)	E (14)	F (15)
0 0 0 0	0	NUL 12-0-9-8-1 NUL 00	DLE 12-11-9-8-1 DLE 10	DS 11-0-9-8-1 80	12-11-0-9-8-1 90	SP no punch SP 20	& 12 & 26	- 11 - 2D	12-11-0 8A	12-0-8-1 a 12-0-1 C3	12-11-8-1 j 12-11-1 CA	~ 11-0-8-1 ~ 11-0-1 D1	12-11-0-8-1 12-11-0-1 D8	{ 12-0 { 12-0 78	\ 11-0 \ 11-0 7D	0-8-2 \ 0-8-2 5C	0 0 0 0 30
0 0 0 1	1	SOH 12-9-1 SOH 01	DC1 11-9-1 DC1 11	SOS 0-9-1 81	9-1 91	12-0-9-1 A0	12-11-9-1 A9	/ 0-1 / 2F	12-11-0-9-1 BB	a 12-0-1 a 61	j 12-11-1 j 6A	~ 11-0-1 ~ 7E	12-11-0-1 09	A 12-1 A 41	J 11-1 J 4A	11-0-9-1 9F	1 1 1 31
0 0 1 0	2	STX 12-9-2 STX 02	DC2 11-9-2 DC2 12	FS 0-9-2 82	SYN 9-2 SYN 16	12-0-9-7 A1	12-11-9-2 AA	11-0-9-2 82	12-11-0-9-2 BC	b 12-0-2 b 62	k 12-11-2 k 6B	s 11 0-2 s 73	12-11-0-7 DA	B 12-2 B 42	K 11-2 K 4B	S 0-2 S 53	2 2 2 32
0 0 1 1	3	ETX 12-9-3 ETX 03	TM 11-9-3 DC3 13	0-9-3 83	9-3 93	12-0-9-3 A2	12-11-9-3 AB	11-0-9-3 83	12-11-0-9-3 BD	c 12-0-3 c 63	l 12-11-3 l 6C	t 11-0-3 t 74	12-11-0-3 DB	C 12-3 C 43	L 11-3 L 4C	T 0-3 T 54	3 3 3 33
0 1 0 0	4	PF 12-9-4 9C	RES 11-9-4 9D	BYP 0-9-4 84	PN 9-4 94	12-0-9-4 A3	12-11-9-4 AC	11-0-9-4 84	12-11-0-9-4 BE	d 12-0-4 d 64	m 12-11-4 m 6D	u 11-0-4 u 75	12-11-0-4 DC	D 12-4 D 44	M 11-4 M 4D	U 0-4 U 55	4 4 4 34
0 1 0 1	5	HT 12-9-5 HT 09	NL 11-9-5 85	LF 0-9-5 LF 0A	RS 9-5 95	12-0-9-5 A4	12-11-9-5 AD	11-0-9-5 85	12-11-0-9-5 BF	e 12-0-5 e 65	n 12-11-5 n 6E	v 11-0-5 v 76	12-11-0-5 DD	E 12-5 E 45	N 11-5 N 4E	V 0-5 V 56	5 5 5 35
0 1 1 0	6	LC 12-9-6 86	BS 11-9-6 BS 08	ETB 0-9-6 ETB 17	UC 9-6 96	12-0-9-6 A5	12-11-9-6 AE	11-0-9-6 86	12-11-0-9-6 C0	f 12-0-6 f 66	o 12-11-6 o 6F	w 11-0-6 w 77	12-11-0-6 DE	F 12-6 F 46	O 11-6 O 4F	W 0-6 W 57	6 6 6 36
0 1 1 1	7	DEL 12-9-7 DEL 7F	IL 11-9-7 87	ESC 0-9-7 ESC 1B	EOT 9-7 EOT 04	12-0-9-7 A6	12-11-9-7 AF	11-0-9-7 87	12-11-0-9-7 C1	g 12-0-7 g 67	p 12-11-7 p 70	x 11-0-7 x 78	12-11-0-7 DF	G 12-7 G 47	P 11-7 P 50	X 0-7 X 58	7 7 7 37
1 0 0 0	8	GE 12-9-8 97	CAN 11-9-8 CAN 18	0-9-8 88	9-8 98	12-0-9-8 A7	12-11-9-8 B0	11-0-9-8 B8	12-11-0-9-8 C2	h 12-0-8 h 68	q 12-11-8 q 71	y 11-0-8 y 79	12-11-0-8 E0	H 12-8 H 48	Q 11-8 Q 51	Y 0-8 Y 59	8 8 8 38
1 0 0 1	9	RLF 12-9-8-1 8D	EM 11-9-8-1 EM 19	0-9-8-1 89	9-8-1 99	12-8-1 A8	11-8-1 81	0-8-1 89	8-1 60	i 12-0-9 i 69	r 12-11-9 r 72	z 11-0-9 z 7A	12-11-0-9 E1	I 12-9 I 49	R 11-9 R 52	Z 0-9 Z 5A	9 9 9 39
1 0 1 0	A (10)	SMM 12-9-8-7 8E	CC 11-9-8-2 92	SM 0-9-8-2 8A	9-8-2 9A	¢ 12-8-2 ¢ 4A	! 11-8-2 ! 5D	: 12-11 : 7C	8-2 3A	12-0-8-2 C4	12-11-8-2 C8	11-0-8-2 D2	12-11-0-8-2 E2	12-0-9-8-2 E8	12 11-9-8-2 EE	11-0-9-8-2 F4	(ILVMI) 12-11-0-9-8-2 FA
1 0 1 1	B (11)	VT 12-9-8-3 VT 0B	CU1 11-9-8-3 8F	CU2 0-9-8-3 8B	CU3 9-8-3 9B	12-8-3 2E	$ 11-8-3 $ 24	0-8-3 2C	# 8-3 # 23	12-0-8-3 C5	12-11-8-3 CC	11-0-8-3 D3	12-11-0-8-3 E3	12-0-9-8-3 E9	12-11-9-8-3 EF	11-0-9-8-3 F5	12-11-0-9-8-3 F8
1 1 0 0	C (12)	FF 12-9-8-4 FF 0C	IFS 11-9-8-4 FS 1C	0-9-8-4 8C	DC4 9-8-4 DC4 14	< 12-8-4 < 4C	' 11-8-4 ' 2A	% 0-8-4 % 25	@ 8-4 @ 40	12-0-8-4 C6	12-11-8-4 CD	11-0-8-4 D4	12-11-0-8-4 E4	∫ 12-0-9-8-4 EA	12-11-9-8-4 F0	⌐ 11-0-9-8-4 F6	12-11-0-9-8-4 FC
1 1 0 1	D (13)	CR 12-9-8-5 CR 0D	IGS 11-9-8-5 GS 1D	ENQ 0-9-8-6 ENQ 05	NAK 9-8-5 NAK 15	(12-8-5 (2B	) 11-8-5) 29	0-8-5 5F	' 8-5 ' 27	12-0-8-5 C7	12-11-8-5 CE	11-0-8-5 D5	12-11-0-8-5 E5	12-0-9-8-5 E8	12-11-9-8-5 F1	11-0-9-8-5 F7	12-11-0-9-8-5 FD
1 1 1 0	E (14)	SO 12-9-8-6 SO 0E	IRS 11-9-8-6 RS 1E	ACK 0-9-8-6 ACK 06	9-8-6 9E	+ 12-8-6 + 2B	¬ 11-8-6 ¬ 3B	> 0-8-6 > 3E	8-6 - 3D	12-0-8-6 C8	12-11-8-6 CF	11-0-8-6 D6	12-11-0-8-6 E6	12-0-9-8-6 EC	12-11-9-8-6 F2	11-0-9-8-6 F8	12-11-0-9-8-6 FE
1 1 1 1	F (15)	SI 12-9-8-7 SI 0F	IUS 11-9-8-7 US 1F	BEL 0-9-8-7 BEL 07	SUB 9-8-7 SUB 1A	| 12-8-7 | 21	¬ 11-8-7 ¬ 5E	? 0-8-7 ? 3F	" 8-7 " 22	12-0-8-7 C9	12 11-8-7 D0	11 0-8-7 D7	12-11-0-8-7 E7	12-0-9-8-7 E0	12-11 9-8-7 F3	11-0 9-8-7 F9	12-11-0 9-8-7 EO FF

LEGEND

REVISION RECORD

REV.	PAGE	PARAGRAPH	DATA	E.C.O.	APP.	DATE
A			Class A Release.			1/80