ETA SYSTEMS

ETA10 Instruction Set Reference Manual

ETA10 Computer System

PUB–1264

A Control Data Company

ETA Systems, Incorporated
1450 Energy Park Drive
St. Paul, MN 55108

ETA SYSTEMS

ETA10 Instruction Set Reference Manual

PUB–1264
Rev. A
March, 1989

ETA10 Computer System

The items listed below are referenced in this document and are names, products, or trademarks associated with the following companies:

ETA is a trademark of ETA Systems, Incorporated.

Disclaimer:

ETA™ Systems, Incorporated reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should consult ETA Systems to determine whether any such changes have been made. (At non-U.S. installations, the reader should consult the local marketing representative.) This manual may not be reproduced and is intended for the exclusive use of ETA Systems' customers.

The terms and conditions governing the sale of ETA Systems hardware products and the licensing and use of ETA Systems software consist solely of those set forth in the written contracts between ETA Systems and its customers. No statement contained in this publication, including statements regarding capacity, suitability for use, or performance of products, shall be considered a warranty by ETA Systems for any purpose or give rise to any liability of ETA Systems.

In no event will ETA Systems be liable for any incidental, indirect, special, or consequential damages (including lost profits) arising out of or relating to this publication or the information contained in it, even if ETA Systems has been advised, knew, or should have known of the possibility of such damages.

Prepared by: ETA Systems, Incorporated
Technical Communication Dept.
1450 Energy Park Drive
St. Paul, MN 55108

Revision Record

Documents that are complete and approved for release carry an alphabetic code. The first release is identified as revision A, the second as revision B, and so on.

Document Revision	Date
Rev. A	March 1989

Changes in This Revision

Revision A is the first release of this manual.

Table of Contents

About This Document xv
Purpose xv
Intended Audience xv
How This Document Is Arranged xv
How to Use This Document xvii
Conventions Used in This Document xvii

Introduction ..... **Chapter 1**

Introduction to the ETA10 Hardware 1-2
Operations Performed by ETA10 Instructions 1-2
ETA10 Instruction Formats 1-3
ETA10 Instruction Functions and Operands 1-3

Introduction to the ETA10 Instruction Set ..... **Chapter 2**

Hardware for Machine Instructions 2-2
Central Processing Units 2-2
Input/Output Units and the Service Unit 2-3
Memories 2-3
Instruction Operations 2-4
Scalar and Vector Operations 2-4
Memory Transfer 2-4
Monitor Operations 2-5
Accessing Special Purpose Registers 2-5
Bit and Byte Operations 2-5
Branching and Indexing 2-5
Floating-Point Arithmetic 2-5
Machine Instruction Formats 2-6
Instruction Function Field 2-12
Instruction Subfunctions 2-12
Subfunctions For Vector Operations 2-14
Control Vectors 2-14
Destination Vector and Control Vector Offsets 2-15

Introduction to the ETA10 Instruction Set ..... Continued

Broadcast Operands 2-15
Sign Control 2-15

ETA10 Instruction Descriptions ..... Chapter 3

Instruction Description Formats 3-2
00 3-3
03 3-3
04 3-4
05 3-6
06 3-6
07 3-7
08 3-8
09 3-9
0A 3-10
0C 3-11
0D 3-11
0E 3-12
0F 3-14
10 3-16
11 3-17
12 3-18
13 3-18
14 3-19
15 3-20
16 3-22
17 3-24
18 3-24
19 3-25
1A 3-25
1B 3-26
1C 3-27
1D 3-27
1E 3-28
1F 3-28
20 3-29
21 3-30
22 3-31
23 3-32
24 3-33
25 3-34

ETA10 Instruction Descriptions ..... Continued

26 3-35
27 3-36
28 3-37
29 3-37
2A 3-38
2B 3-38
2C 3-39
2D 3-39
2E 3-40
2F 3-41
30 3-42
31 3-43
32 3-44
33 3-46
34 3-48
35 3-49
36 3-50
37 3-52
38 3-52
39 3-53
3A 3-53
3B 3-54
3C 3-54
3D 3-55
3E 3-55
3F 3-56
40 3-56
41 3-57
42 3-57
44 3-58
45 3-58
46 3-59
48 3-59
49 3-60
4B 3-60
4C 3-61
4D 3-61
4E 3-62
4F 3-62
50 3-63
51 3-64
52 3-65

ETA10 Instruction Descriptions ..... Continued

53 3-66
54 3-67
55 3-68
56 3-69
57 3-71
58 3-72
59 3-72
5A 3-73
5B 3-73
5C 3-74
5D 3-75
5E 3-76
5F 3-76
60 3-77
61 3-77
62 3-78
63 3-78
64 3-79
65 3-79
66 3-80
67 3-80
68 3-81
69 3-81
6B 3-82
6C 3-82
6D 3-83
6E 3-83
6F 3-84
70 3-85
71 3-86
72 3-87
73 3-88
74 3-89
75 3-90
76 3-91
77 3-92
78 3-93
79 3-93
7A 3-94
7B 3-94
7C 3-95
7D 3-96

ETA10 Instruction Descriptions ..... Continued

7E 3-97
7F 3-97
80 3-98
81 3-99
82 3-100
83 3-101
84 3-102
85 3-103
86 3-104
87 3-105
88 3-106
89 3-107
8A 3-108
8B 3-109
8C 3-110
8F 3-111
90 3-112
91 3-113
92 3-114
93 3-115
94 3-116
95 3-118
96 3-120
97 3-121
98 3-122
99 3-123
9A 3-124
9B 3-125
9C 3-126
9D 3-128
A0 3-130
A1 3-132
A2 3-134
A4 3-136
A5 3-138
A6 3-140
A8 3-142
A9 3-144
AB 3-146
AC 3-148
AF 3-150
B0 3-152

ETA10 Instruction Descriptions ..... Continued

B1 3-154
B2 3-156
B3 3-158
B4 3-160
B5 3-162
B0 3-164
B1 3-166
B2 3-168
B3 3-170
B4 3-172
B5 3-174
B0 3-176
B1 3-177
B2 3-178
B3 3-179
B4 3-180
B5 3-181
B0 3-182
B1 3-183
B2 3-184
B3 3-185
B4 3-186
B5 3-187
B6 3-188
B7 3-189
B8 3-191
BA 3-192
BB 3-193
BC 3-194
BD 3-195
BE 3-196
BF 3-196
C0 3-197
C1 3-198
C2 3-199
C3 3-200
C4 3-201
C5 3-202
C6 3-203
C7 3-204
C8 3-205
C9 3-206

ETA10 Instruction Descriptions ..... Continued

CA 3-207
CB 3-208
CC 3-209
CD 3-210
CE 3-210
CF 3-211
D0 3-212
D1 3-213
D4 3-214
D5 3-215
D8 3-216
D9 3-218
DA 3-220
DB 3-221
DC 3-222
DF 3-223
F0 3-224
F1 3-225
F2 3-226
F3 3-227
F4 3-228
F5 3-229
F6 3-230
F7 3-231
F8 3-232
FA 3-233
FB 3-235
FC 3-237
FD 3-238
FE 3-239
FF 3-239

Appendix A: Instructions by Function Code A-1

Appendix B: Instructions By Mnemonic B-1

Appendix C: Instructions With Sign Control C-1

Appendix D: Instructions With Broadcasting D-1

Appendix E: Instruction Termination Rules E-1

Appendix F: Floating-Point Operations F-1

Floating-Point Format F-1
Two's Complement Notation F-2
Floating-Point Arithmetic F-4
Right Normalization F-5
Floating-Point Addition F-6
Floating-Point Subtraction F-8
Floating-Point Multiplication F-10
Floating-Point Division F-12
Normalized Upper Results F-14
Double-Precision Results F-15
Floating-Point Square Root Operations F-16
Significant Results F-17
Floating-Point Comparison Rules F-18
Indefinite Operand(s) F-18
Machine Zero Operand(s), Not Indefinite F-18
Operand(s) Not Indefinite or Machine Zero F-19

Appendix G: The Data Flag Register G-1

Data Flag Register Format G-1
Data Flags G-2
The Mask Field G-2
Product Field Bits G-2
Data Flag Branch Enable Bit G-2

Causing a Data Flag Branch G-3
Data Flag Register Bit Assignments G-4
Free Data Flags G-5
Instructions Affecting Data Flag Register Bits G-6

Appendix H: Addressing Vector Operands H-1

Addressing Vector Source Operands H-1
Source Operand Offsets H-1
Addressing Vector Result Operands H-2
Result Operand Offsets H-2

Appendix I: Illegal Instructions I-1

Type One Illegal Instructions I-1

Type Two Illegal Instructions I-1

Glossary Glossary-1

Referenced Documents List Ref.Doc.-1

Index ..... Index

List of Figures

Figure	*Title*	*Page*
2-1.	Components of an ETA10 central processing unit (CPU).	2-2
3-1.	Structure of Register R for the #04 Instruction.	3-5
3-2.	Structure of Register R for the #08 Instruction.	3-8
3-3.	Register T after an #0E instruction.	3-12
3-4.	Structure of Register R for the #0F instruction.	3-15
3-5.	Bit Compress Operations.	3-19
3-6.	Bit Merge Operations.	3-20
3-7.	Bit Mask Operations.	3-22
F-1.	A 32-Bit Floating-Point Number's Format.	F-1
F-2.	A 64-Bit Floating-Point Number's Format.	F-2
F-3.	Floating-Point Result Format for Add, Subtract, and Multiply Operations.	F-4
F-4.	Floating-Point Addition.	F-7
F-5.	To perform floating point subtraction, complement the subtrahend, then add.	F-9
F-6.	Floating-Point Multiplication.	F-11
F-7.	Floating-Point Division.	F-13
G-1.	Data Flag Register Format.	G-1

List of Tables

Table	*Title*	*Page*
2-1.	Instruction Qualifiers.	2-13
2-2.	Qualifiers and valid G-bit settings for vector operations.	2-14
3-1.	Vector instructions that can be used in a Link operation.	3-69
3-2.	Valid Combinations for Linked Vector Instructions.	3-70
3-3.	Logical Operations on vector A and B elements.	3-128
3-4.	Logical Functions on X and Y to Produce Order Vector Z.	3-130
3-5.	Results of the logical operations performed by the source vectors.	3-131
3-6.	Logical Functions on X and Y to Produce Order Vector Z.	3-132
3-7.	Results of the logical operations performed by the source vectors.	3-133
3-8.	Logical Functions on X and Y to Produce Order Vector Z.	3-134
3-9.	Results of the logical operations performed by the source vectors.	3-135
3-10.	Logical Functions on X and Y to Produce Order Vector Z.	3-136
3-11.	Results of the logical operations performed by the source vectors.	3-137
3-12.	Logical Functions on X and Y to Produce Order Vector Z.	3-138
3-13.	Results of the logical operations performed by the source vectors.	3-139
3-14.	Logical Functions on X and Y to Produce Order Vector Z.	3-140

Table	*Title*	*Page*
3-15.	Results of the logical operations performed by the source vectors.	3-141
3-16.	Logical Functions on X and Y to Produce Order Vector Z.	3-142
3-17.	Results of the logical operations performed by the source vectors.	3-143
3-18.	Logical Functions on X and Y to Produce Order Vector Z.	3-144
3-19.	Results of the logical operations performed by the source vectors.	3-145
3-20.	Logical Functions on X and Y to Produce Order Vector Z.	3-146
3-21.	Results of the logical operations performed by the source vectors.	3-147
3-22.	Logical Functions on X and Y to Produce Order Vector Z.	3-148
3-23.	Results of the logical operations performed by the source vectors.	3-149
3-24.	Logical Functions on X and Y to Produce Order Vector Z.	3-150
3-25.	Results of the logical operations performed by the source vectors.	3-151
A-1.	Instructions by Function Code	A-1
B-1.	Instructions by Mnemonic	B-1
C-1.	Instructions for which sign control is valid.	C-1
D-1.	Instructions Allowing Broadcasting.	D-1
E-1.	Instruction Terminating Conditions	E-1
G-1.	Data Flag Bit Settings	G-4
G-3.	Definitions For Free Data Flag Bits 53-55.	G-6
G-4.	Data Flag Bits set by function codes.	G-7
I-1.	The Domain Package's Illegal Instruction Mask	I-2

About This Document . . .

Purpose

This document is a reference manual for the ETA10 instruction set. It is **not** intended for use as a guide for assembly language programmers. (Refer to the **Referenced Documents List** for the title of manuals that contain assembly language information.) The manual is designed to provide quick access to reference information about each instruction's format and function.

Refer to PUB–1005, *ETA10 System Reference Manual*, for detailed information about ETA10 operations.

Intended Audience

The manual's audience includes:

- Programmers in higher-level languages such as FORTRAN, C, and CYBIL, who may be reading assembler output from programs
- Programmers who may be writing Q8 calls in FORTRAN programs
- Site analysts
- On-site engineers

How This Document Is Arranged

There are 3 chapters and 9 appendices in this document.

Chapter 1, Introduction, covers the manual's contents, summarizing the information presented in each chapter.

Chapter 2, Introduction to the ETA10 Instructions, gives an overview of the ETA10 hardware, and summarizes the types of operations the machine can perform. The thirteen instruction formats are laid out, with an explanation of designator meanings for each format. Instruction subfunctions are also described.

Chapter 3, Instruction Descriptions, lists each instruction in hexadecimal function code order. The description covers the

instruction's function, valid qualifiers and G-bit settings, with an explanation of the operations performed. See Appendix B for a listing of the instructions in mnemonic order.

Appendix A, Instructions by Function Code, contains each instruction organized by function code, with its mnemonic, format type, the G-bit settings, and a brief definition of its operation.

Appendix B, Instructions by Mnemonic, contains each instruction organized by its mnemonic, function code, format type, the G-bit settings, and a brief definition of its operation.

Appendix C, Instructions Using Sign Control, is a table of instructions for which sign control is valid. The list is organized by function code. The sign control G-bits valid for each instruction are included in the list.

Appendix D, Instructions Allowing Broadcasting, lists, by function code, those instructions that can have broadcast A or B operands. The entry for each instruction includes whether A, B, or both can be broadcast.

Appendix E, Instruction Terminating Conditions, lists the terminating conditions for instructions, depending on their operands.

Appendix F, Floating-Point Operations, discusses how floating-point arithmetic is performed on the ETA10.

Appendix G, Data Flag Register Bit Settings, describes the function and format of the data flag register, with the meanings of bit settings that cause branching.

Appendix H, Vector Operands, explains how designators on vector instructions refer to registers that address source and destination vectors, and may specify offsets.

Appendix I, Illegal Instructions, describes which instructions are illegal and the consequences of issuing illegal instructions.

The **Glossary** provides definitions of important terms found in this manual.

How to Use This Document

The information provided in this manual assumes that the reader is familiar with the information in PUB-1005, *ETA10 System Reference Manual.*

For an overview of topics presented in this manual, read chapter one. Read chapter two for details about instruction formats and designator meanings, and for information about instruction subfunctions, particularly for instructions that use sign control.

To find information about how a specific instruction operates, refer to the instruction's description in chapter three.

See the appendices for tables summarizing certain instruction characteristics, and for subjects referred to in the instruction descriptions, such as floating-point operations.

Conventions Used in This Document

Numbers that are represented in hexadecimal format in the text have a pound sign (#) as prefix.

The mnemonics used throughout this manual are those of the ETA System V assembler, "as".

Chapter 1

Introduction

In This Chapter . . .

Chapter one introduces topics about the ETA10 and its instruction set that are covered in chapters two and three and the appendices of this manual. This chapter contains the following sections:

- Introduction to the ETA10 Hardware
- Operations Performed by ETA10 Instructions
- ETA10 Instruction Formats
- ETA10 Instruction Functions and Operands

Introduction to the ETA10 Hardware

The ETA10 is a multi-processor system consisting of Central Processor Units (CPU), Input-Output Units (IOU), the Service Unit, and a hierarchical memory.

IOUs are processors responsible for data movement through the system. The Service Unit allows operators to interact with the ETA10, monitoring and controlling its functions.

CPUs interpret and execute instructions in the system. Each CPU has scalar and vector processors, 256 general purpose registers, and its own central processor (CP) memory.

The hierarchical memory system consists of three memories: shared memory (SM), the communication buffer (CB), and CP memory (CPM). Shared memory is a large auxiliary storage area for CP memory data, accessible from each CPU. Each CPU has its own CP memory, holding machine instructions and data. The communication buffer, a fast memory used for high-speed synchronization messages and semaphore operations, is accessible from each CPU.

The section "Hardware For Machine Instructions" in chapter 2 briefly describes the system components.

Operations Performed by ETA10 Instructions

There are 216 hardware instructions performing:

- Scalar and vector operations
- Memory transfers
- Monitor operations
- Access to special purpose registers
- Bit and byte manipulation
- Branching and indexing
- Floating-point arithmetic

The section "Instruction Operations" in chapter 2 expands on these topics.

ETA10 Instruction Formats

Instructions are 32 or 64 bits long. There are 13 instruction formats, six of which are 64-bit instructions. The other seven formats are 32 bits long.

Each instruction word is divided into fields, bit groups that have instruction designators defining the function and operands. Each designator field is usually 8 bits long; some formats have designators that are longer.

The instruction formats and designator descriptions are laid out in the chapter 2 sections "Machine Instruction Formats" and "Instruction Designators".

ETA10 Instruction Functions and Operands

All instructions have a function code, a number from #00 through #FF, that describes the operation performed. An instruction performs its function on operands. The number, format, and meaning of instruction operands depend on each instruction format.

Many instructions have an 8-bit subfunction field that further defines the function. For example, instructions performing vector operations have a subfunction field describing: operand size, whether a control vector acts on zeros or ones, the offset applied to the output field, whether operands are broadcast, and what sign control is valid.

"Instruction Command Field" in chapter 2 provides details about the function and subfunction fields.

Chapter 2

Introduction to the ETA10 Instruction set

In This Chapter . . .

Chapter two introduces the instruction set in terms of:

- Hardware for Machine Instructions
- Instruction Operations
- Machine Instruction Formats
- Instruction Designators
- Instruction Function Field
- Instruction Subfunctions

Hardware for Machine Instructions

The ETA10 is a multiprocessor computer system, all processors having access to a large shared memory. All peripheral and network connections are through I/O units into shared memory. The components of an ETA10 central processing unit (CPU) are shown in figure 2-1, and introduced in the following sections. Refer to PUB-1005, *ETA10 System Reference Manual*, for a more detailed discussion of the ETA10 components.

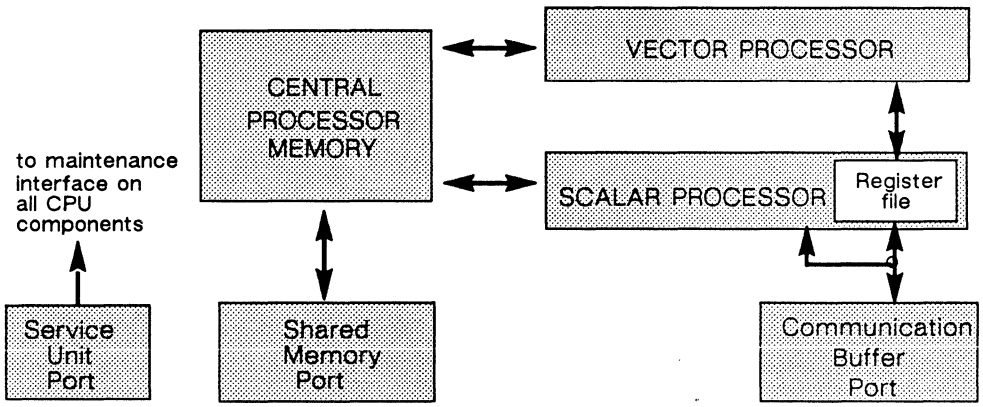


Figure 2-1. Components of an ETA10 central processing unit (CPU).

Central Processing Units

A central processor unit (CPU) is the functional unit that interprets and executes instructions in the system. Each CPU has a central processor that operates independently, with its own scalar and double-pipelined vector processors, 256 general purpose 64-bit registers, and CP memory.

Each CPU is directly connected to shared memory and communication buffer ports for data transfers, and to the communication buffer for communication with other system processors. Maintenance Interface logic on each component allows the Service Unit to perform diagnostic and maintenance functions on each CPU.

A CPU runs in Job or Monitor mode. Modes change when the #09 exchange instruction executes. Some operating characteristics change, depending on the new mode. In Monitor mode, memory is physically addressed, register #03 points to the next branch instruction, and a #09 exchange to Job mode instruction is the last instruction executed. Job mode addresses memory virtually, the Invisible Package holds the next branch instruction, and any instruction can be the last executed before an exchange to Monitor mode occurs.

Input/Output Units and the Service Unit

An Input/Output Unit (IOU) is a specialized multi-processor, bus-connected computer system that contains a set of channel processors, 2 SIO lines, a data pipe controller, and global memory. IOUs are responsible for all data movement through the system to peripherals (including networks). They provide a means to attach peripheral devices and networks. A super-cooled ETA10 supports up to 18 IOUs.

The Service Unit (SU) provides access for operator display and control, system reconfiguration, and maintenance functions.

Memories

The ETA10's hierarchical memory system consists of CP memory, shared memory, and the communication buffer.

Each CPU has its own CP memory that holds machine instructions and data. CP memory is accessible by its central processor and the service unit, and under direction of the CPU, data can be transferred between CP memory and shared memory.

CP memory can be addressed two ways, virtually and physically. Virtual storage is divided into regions with contiguous address called 'pages.' Each page is identified by a unique virtual page address, and is associated with a unique physical page address while in CP memory.

Shared memory provides large bulk auxiliary storage for CP memory data. In super-cooled systems, access is via the shared memory interface (SMI) that supports up to 8 high-speed CPU ports, and 20 low-speed ports for IOU and SU connections. Data is transferred in blocks in half-word or full-word transfer units, ranging from a half-word to 64K words.

The communication buffer (CB) offers fast auxiliary storage, and is used to transmit high-speed synchronization messages and signals among the system components. In a super-cooled sytem, it can be divided into halves. Each half has its own interface that connects CB to 10 ports supporting up to 8 CPUs and the system's IOUs. Base/Limit/Access Pairs (BLAPs) denote the lowest numbered (base) and highest numbered (limit) CB address accessible by a CPU, and the operations that the CPU can perform on a range of CB addresses (access rights). The BLAPs are defined in domains in the CPU. Each domain has a set of 4 BLAPs, and can permit access to up to 4 ranges of CB memory at once.

Instruction Operations

The ETA10's 216 instructions are model-independent. Instructions #0*x* through #7*x* are 32-bits long, and instructions #8*x* through #F*x* are 64-bits long. The CPU's register file has registers that are available to the instructions as a source of operands, and as a destination for the result. Instructions perform a variety of operations; the main ones are summarized in this section.

Scalar and Vector Operations

Designators in scalar instructions point to registers that are sources and destinations. Registers contain the source operands and results.

The emphasis of the ETA10 is on vector operations. Vector instructions process vectors that stream data from source to destination locations in CP memory. Instruction designators point to registers that describe the sources and destinations; the sources and destinations are usually vectors, not single quantities. Vector instructions address vector operands and control how results are stored. Qualifiers modify the instruction's function.

Memory Transfer

All central processors can access the communication buffer to synchronize and coordinate system-wide programs. Instructions perform word and half word transfers between CB and the register file, semaphore post and wait operations, conditional word/half-word swap from CB to the register file, and conditional test and set with word/half word load from CB to register file.

Shared memory instructions manage data transfer between CP memory and shared memory by setting up a queue of information to transfer. Instructions build Transfer Request Blocks (TRBs) describing the type of transfer, set up an input queue in CP memory, and place TRBs awaiting execution in the input queue. The hardware reads TRBs off the input queue, and transfers the data until the queue is exhausted. After a TRB is read from CP memory, it may be placed in a completion queue residing in CP memory.

Shared memory instructions check for the transfer's completion status, and can also stop and restart I/O between TRBs to allow the input queue to be adjusted.

Monitor Operations

Instructions are available to perform privileged monitor operations unavailable in Job mode. These instructions function in Monitor mode only. Their operations include address translation, loading and storing associative registers, loading keys, and loading the Monitor Interval Timer.

Accessing Special Purpose Registers

Access to special purpose registers such as the Real-Time Clock, the Job Interval Timer, the Monitor Interval Timer, and the Breakpoint Register is possible using instructions. An important special purpose register is the Data Flag Register, which provides for status conditions, and causes an automatic branch to a special routine upon encountering certain operands, results, or conditions.

Bit and Byte Operations

Data can be manipulated by instructions at the bit and byte level. Bytes can be moved, loaded, and stored. Bit streams may be compressed, merged, masked, counted, and logically processed.

Branching and Indexing

Execution can proceed elsewhere in a program unconditionally or based on the result of a comparison. Single bit, 24- or 48-bit integer, 32- or 64-bit integer, 32- or 64-bit floating-point operands can be compared.

Special branching occurs when the #09 Exit Force Instruction passes control between Monitor and Job mode programs. The #36 (Branch or Forward Domain Change), and #17 (Backward Domain Change) instructions go between different domains of a job program.

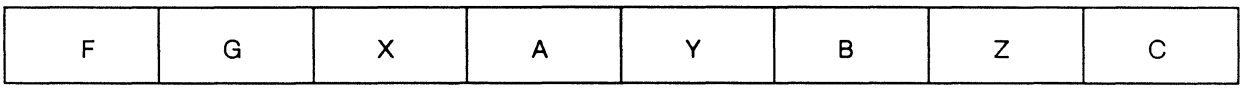
Indexing is applied to addressing to load and store instructions, branch instructions, and string instructions.

Floating-Point Arithmetic

Instructions perform floating-point arithmetic on 32- or 64-bit floating-point numbers, returning upper, lower, normalized, and significant results. Numbers may also be compared according to floating-point comparison rules. Several instructions produce double-precision results. Appendix F explains floating-point arithmetic in detail.

Machine Instruction Formats

The ETA10 instructions have thirteen formats, numbered #1 through #D. Six formats are 64 bits long, and seven are 32 bits long. Each format is divided into bit groups that have assigned instruction designators. The thirteen formats are described below, with their designators labeled by letters (F, G, X, A etc.). Shaded areas are unused. The meaning of each designator is listed. All fields are 8 bits long unless otherwise specified.



Format #1

F Function code of instruction.

G An 8-bit designator that specifies certain subfunction conditions. Subfunctions include length of operands (32- or 64-bit), normal or broadcast source vectors, and so on. The number of bits used in the G designator varies with instructions.

X Specifies a register that contains the offset or index for vector or string source field A.

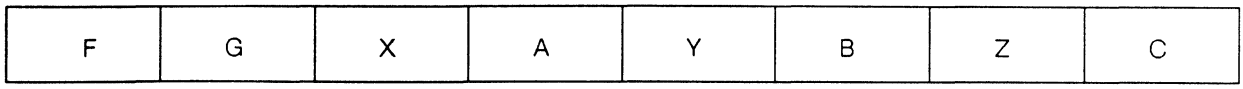
A Specifies a register that contains a field length and base address for the corresponding source vector or string field.

Y Specifies a register that contains the offset or index for vector or string field B.

B Specifies a register that contains a field length and base address for the corresponding source vector or string field.

Z Specifies a register that contains the base address for the order vector used to control the result vector in field C.

C Specifies a register that contains the field length and base address for storing the result vector or string field. C+1 specifies a register containing the offset for C and Z vector fields. If the C+1 designator is used, the C designator must specify an even-numbered register.



Format #2

F Function code of instruction.

G An 8-bit designator that specifies certain subfunction conditions. Subfunctions include length of operands (32- or 64-bit), normal or broadcast source vectors, and so on. Number of bits used in the G designator varies with instructions.

X Specifies a register that contains length and base address for order vector corresponding to source sparse vector field A.

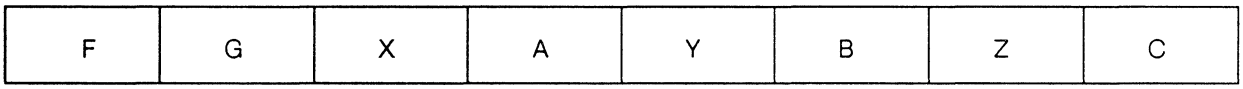
A Specifies register containing the base address for a source sparse vector field.

Y Specifies a register that contains the length and base address for the order vector corresponding to source sparse vector field B.

B Specifies register containing the base address for a source sparse vector field.

Z Specifies a register that contains the length and base address for the order vector corresponding to result sparse vector field C.

C Specifies a register that contains the field length and base address for storing the result vector or string field.



Format #3

F Function code of instruction.

G An 8-bit designator that specifies certain subfunction conditions. Subfunctions include length of operands (32- or 64-bit), normal or broadcast source vectors, and so on. The number of bits used in the G designator varies with instructions. For some format 3 instructions, the G designator is used as an immediate byte I8.

X Specifies a register that contains the offset or index for vector or string source field A.

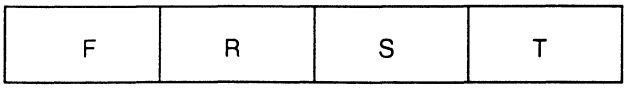
A Specifies a register that contains a field length and base address for the corresponding source vector or string field.

Y Specifies a register that contains the offset or index for vector or string field B.

B Specifies a register that contains a field length and base address for the corresponding source vector or string field.

Z Specifies a register that contains the index for result field C.

C Specifies a register that contains the field length and base address for storing the result vector or string field.



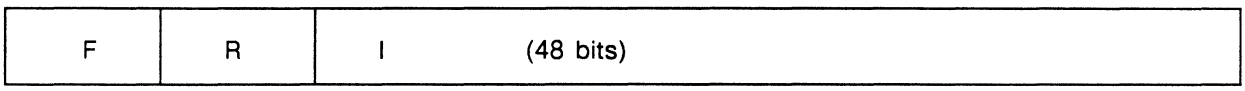
Format #4

F Function code of instruction.

R Specifies a register containing an operand for use in an arithmetic operation.

S Specifies a register containing an operand for use in an arithmetic operation.

T Specifies a destination register for the transfer of the arithmetic results.

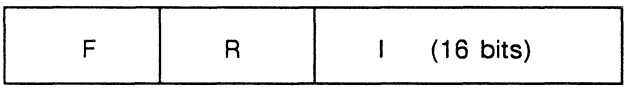


Format #5

F Function code of instruction.

R Specifies a destination register for the transfer of an operand or operand sum.

I 48-bit index used to form the branch address in a #B6 branch instruction. In #BE and #BF index instructions, I is a 48-bit operand.

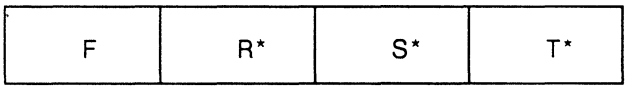


Format #6

F Function code of instruction.

R Specifies a destination register for the transfer of an operand or operand sum.

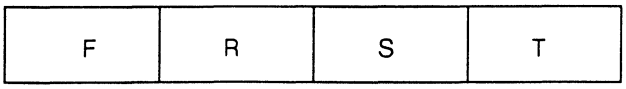
I A 16-bit operand.



Format #7

F Function code of instruction.

* Described where used.



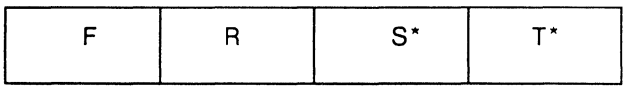
Format #8

F Function code of instruction.

R Specifies registers and branching conditions given in the individual instruction descriptions.

S Specifies registers and branching conditions given in the individual instruction descriptions.

T Specifies a register that contains the base address and, in some cases, the field length of the corresponding result field or branch address.

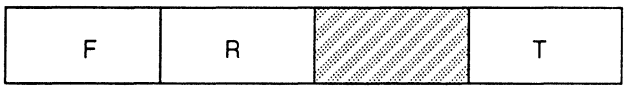


Format #9

F Function code of instruction.

R Specifies registers and branching conditions given in the individual instruction descriptions.

* Described where used.

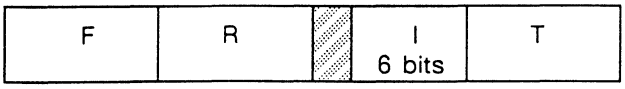


Format #A

F Function code of instruction.

R Specifies registers and branching conditions given in the individual instruction descriptions.

T Specifies a register containing the old state of a register, DFB register, and so on; in an index, branch, or inter-register transfer operation.



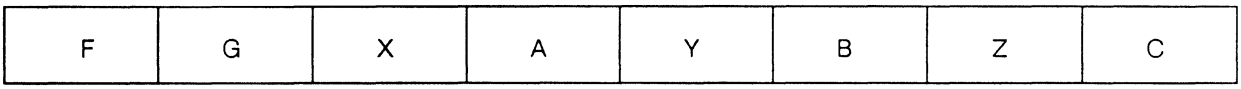
Format #B

F Function code of instruction.

R Specifies registers and branching conditions given in the individual instruction descriptions.

I In the #33 branch instruction, the 6-bit I is the number of the DFB object bits used in the branching operation.

T Specifies a register containing the old state of a register, DFB register, and so on; in an index, branch, or inter-register transfer operation.



Format #C

F Function code of instruction.

G An 8-bit designator that specifies certain subfunction conditions. Subfunctions include length of operands (32- or 64-bit), normal or broadcast source vectors, and so on. The number of bits used in the G designator varies with instructions.

X Specifies a full word or half word register that contains an operand, the length and type of which is determined by G field bits.

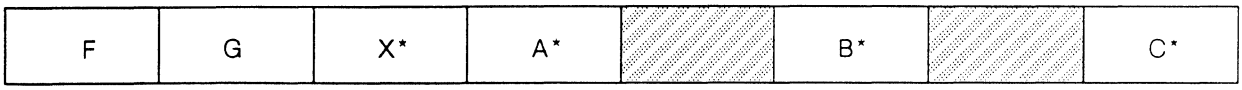
A Specifies a full word or half word register, the length and type of which is determined by G field bits.

Y Specifies one of the following: a register that contains an index used to form the branch address; part of the half word item count in a relative branch; or a destination register for storing a one if the condition is met, and zero otherwise.

B Specifies a register that contains the branch base address in the rightmost 48 bits, or must be set to zero, depending on G bit 2.

Z Contains a two's complement or unsigned integer that determines whether the condition is met.

C Specifies a full word or half word register that contains the sum of (A) + (X) for indexed branch instructions, but must be set to zero for compare floating-point instructions.



Format #D

F Function code of instruction.

G An 8-bit designator that specifies certain subfunction conditions. Subfunctions include length of operands (32- or 64-bit), normal or broadcast source vectors, and so on. The number of bits used in the G designator varies with instructions.

* Described where used.

Instruction Function Field

Each instruction has a function and most have operands. All instructions have a function field, the first byte in the instruction. The function value ranges from #00 through #FF. #00 through #7F are 32-bit instructions, and #80 through #FF are 64–bit instructions. The function defines the operation that the instruction performs.

The instruction performs its function on operands, and the operands' number, format, and meaning depend on the instruction. In the instruction word, operands are generally defined by 8-bit designators that are translated into register numbers, address offsets and bases, and immediate data.

Instruction Subfunctions

Many machine instructions have a subfunction field (referred to as the "G-field"), which further defines the instruction's function. Bits 0–7 in the G field ('G–bits' 0–7) are set to determine the subfunction. Bit setting meanings may vary, depending on the instruction. (Note that in some instructions, #56 for example, the R–field contains subfunctions and uses 'G–bits.')

Table 2–1 lists the qualifier mnemonics used with the instructions in this manual, their hexadecimal values, and the qualifier's meaning. Note that the hexadecimal values listed in the table must be added when more than one qualifier is specified for an instruction.

For the convenience of program developers, the qualifier associated with each of the G–bits is also included in the table. The description for each instruction in this manual refers to G–bit usage by means of these qualifiers. The absence of a qualifier in an instruction description means that the corresponding G–bit must be a zero; the presence of a qualifier means that the corresponding G–bit must be a one.

The first digit of the value is the hexadecimal value of G–bits 0–3, the second digit is the hexadecimal value of G–bits 4–7. For example, qualifier *rel* has a hexadecimal values of 0 and 4. The bit settings are then 00000100.

Table 2–2 lists the G–bit definitions associated with most vector instructions.

Table 2-1. Instruction Qualifiers.

Qualifier	#Value	G-bits Set	Meaning
a	10	3	Broadcast A operand
b	08	4	Broadcast B operand
br	40	1	Unconditional branch
brb	06	5 & 6	Relative branch backward
brf	04	5	Relative branch forward
bro	80	0	Branch on one
brz	C0	0 & 1	Branch on zero
c	02	6	Complement A operand
ca0	00	none	CB address base, limit, access select 0
ca1	01	7	CB address base, limit, access select 1
ca2	02	6	CB address base, limit, access select 2
ca3	03	6 & 7	CB address base, limit, access select 3
fia	04	5	Use fixed increment A
fwc	10	3	Full word boolean compare (64 bits)
grp	02	6	Transmit elements in groups
ivg	60	1 & 2	Implication vector generation
h	80	0	Half word operand
lh	20	2	Start at last hit
ma	04	5	Magnitude of A operand
mb	01	7	Magnitude of B operand
n	06	5 & 6	Negative A operand
neq	01	7	Search for inequality
o	20	2	Offset destination and control vector
pa0	00	none	CB process word address base, limit, access select 0
pa1	01	7	CB process word address base, limit, access select 1
pa2	02	6	CB process word address base, limit, access select 2
pa3	03	6 & 7	CB process word address base, limit, access select 3
ra	10	3	First operation's result replaces A input to second operator
rb	08	4	First operation's result replaces B input to second operator
rel	04	5	Relative branch (forward or backward)
rf	01	7	Source/destination resides in the register file
rvg	20	2	Reverse vector generation
sa0	00	none	CB semaphore address base, limit, access select 0
sa1	10	3	CB semaphore address base, limit, access select 1
sa2	20	2	CB semaphore address base, limit, access select 2
sa3	30	2 & 3	CB semaphore address base, limit, access select 3
sb	01	7	Skip B on each A stored
sc	20	2	Set condition
so	20	2	Set bit to one
sz	30	2 & 3	Clear bit to zero
t	10	3	Toggle bit
usi	08	4	Use 48-bit unsigned integers
xvg	40	1	Exclusive OR vector generation
z	40	1	Control vector on zeros

Subfunctions For Vector Operations

Vector instructions all have an 8-bit G-field. The G-field bit settings for a particular instruction affect its operand size, how the control vector operates, whether operands are broadcast, and if there is any sign control. Table 2–2 shows only the qualifiers used with vector instructions, the G-bits set by each qualifier, and the meaning. Explanations of the different subfunctions follow the table. Refer to table 2–1 for a complete list of instruction qualifiers.

Table 2–2. Qualifiers and valid G-bit settings for vector operations.

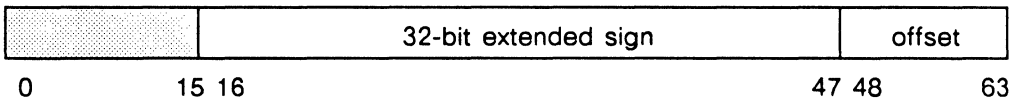
Qualifier	G-bit	State	Meaning
h	0	0	Operands are 64 bits long (word)
		1	Operands are 32 bits long (half word)
z	1	0	Control vector operates on binary ones
		1	Control vector operates on binary zeros
o	2	0	No offset for destination field and control vector
		1	Offset for destination field and control vector
a	3	0	Vector A is the source operand
		1	Broadcast repeated constant in register A
b	4	0	Vector B is the source operand
		1	Broadcast repeated constant in register B
ma,mb, c,n	5,6,7	0	Sign control

Control Vectors

The Z designator specifies a register containing the control vector's base address. A control vector is a bit vector. Each bit is associated with storing a result in the corresponding element of the destination vector. If a control vector is specified in an instruction (non-zero Z designator), the *z* qualifier can be used to set bit 1 of the G-field. *z* determines whether a zero or one control vector bit allows a result to be stored. Data flag bits are set only for operands that are stored. If *z* is specified, the result is stored if the corresponding bit in the control vector is zero. Otherwise, the result is stored if the control vector bit is set to one. A Z designator of zero causes all result elements to be stored in the destination field without regard to the *z* qualifier. The control vector uses the same length used by the destination field.

Destination Vector and Control Vector Offsets

The C designator specifies a register containing the destination vector's field length and base address. If the *o* qualifier is specified (setting G-bit 2), register C+1 contains an offset into the destination. The same offset applies to the control vector. The format of register C+1 is:



If an offset is specified, C must be specified as an even number. The offset is added to the base address to get the destination and control vector starting addresses (it is a bit offset for the control vector). The offset is an item count. Before being added to the base address, the offset is multiplied by a factor adjusting for the size of the operands. It is shifted left six places for 64-bit operands, and five places for 32-bit operands. The offset is subtracted from the field length specified for the destination field. A C designator of zero has no destination field. Note that offsets also apply to input vectors, refer to Appendix H.

Broadcast Operands

Qualifiers *a* and *b* control the setting of G-bits 3 and 4 that define broadcasts for the A and B source operand streams. If G-bits 3 and 4 are not set, vectors A and B from CPU memory are the sources. If they are set, the A or B source field is a constant obtained from the respective register, a repeated operand that is broadcast for the length of the operation. The constant becomes each element of the A or B vector stream. If the *h* qualifier is set to define 32-bit operands, the source operand registers are 32–bit registers. Some instructions do not permit the use of one or both of the *a* and *b* qualifiers.

Registers A and B contain the field length and base address of the two source operand streams when broadcast is not specified. Registers X and Y, respectively, contain the offsets. The offsets modify the field length and base address of the source fields just as described for the destination field. A non–broadcast source field that is shorter than the destination field is extended with operands (as described in Appendix E).

Sign Control

On some vector operations, G-bits 5, 6, and 7 are used to define sign control for input operands. Four qualifiers – *c, ma, mb,* and *n* – control the state of the three G bits. If no qualifiers are set, vector A and B stream operands are used in the normal way.

The *c* qualifier sets G-bit 6 to complement the coefficients of the A stream operands before they are used. The *ma* qualifier sets G-bit 5 to use the magnitude of the coefficients of A stream operands. *mb* sets G-bit 7 to use the magnitude of the coefficients of B stream operands. The *n* qualifier may be used only if neither *ma* nor *mb* is specified. *n* sets bits 5 and 6 to use the negative form of A operands; all positive coefficients of the A stream operands are complemented before being used. Negative operands are not changed.

Appendix C lists the instructions for which sign control is valid.

Chapter 3

ETA10 Instruction Descriptions

In This Chapter . . .

The ETA10 instruction descriptions are listed in hexadecimal order of the function code. They include the instruction format, G-bit settings and qualifier mnemonics, as well as a short description of the operation.

Instruction Description Formats

The instruction descriptions in this chapter occupy one-half, one, or two pages. Instruction are arranged by their numeric function code, #00 through #FF.

The description includes the instruction's:

- Length (half word or full word).
- Format (#1 through #D).
- Subfunction and qualifiers, if applicable. In the example below, all subfunction bits in the G-field may be set. The valid qualifiers are *h,z,o,a,b,ma,c,n,* and *mb*.

  Subfunction: hzoabsss
  Qualifiers: h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]

- Instruction word layout, showing the designators (F,G,A,B, and so on). Shading indicates unused areas. (Unused areas of an instruction must always be cleared to zero.)
- Operations performed. A brief discussion of how the instruction functions, with any resulting data flag bit settings. (Some string and vector macro instructions that return a result to the register file, and the Data Flag Register, do not alter the location in register file nor the Data Flag Register if the instruction is a no-op.)

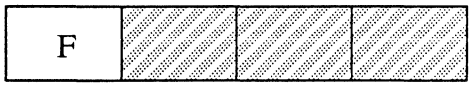
Hexadecimal numbers are prefixed by a pound sign (#).

For information about instruction operations mentioned in the descriptions, refer to chapter two and the appendices of this manual. Refer to PUB-1005, *ETA10 System Reference Manual*, for information about such topics as Job and Monitor mode, domain changes, virtual and physical addressing, and so on.

00

Idle

Half Word, Format #7
Subfunction: None

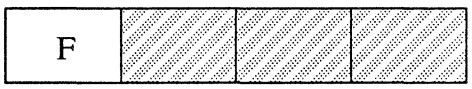


The #00 instruction is used in Monitor mode. The idle is terminated when an interrupt occurs. When this happens, the instruction branches to the absolute half word address in register #03. The Trace Register is entered with this instruction's address when the branch occurs.

03

No Operation

Half Word, Format #7
Subfunction: None



This instruction is a no-op.

04

Breakpoint On Address

Half Word, Format #4
Subfunction: None

The #04 instruction transfers to the breakpoint register the contents of the 64-bit register designated as R. The breakpoint register is a maintenance and programming debugging aid.

The breakpoint function compares addresses of specified categories of requests with the address in the breakpoint register. In Job mode, virtual addresses are compared; in Monitor mode, absolute addresses. Breakpoint compares are disabled for the absolute addressing of CP memory by exchanges, domain changes, space table searches, and shared memory transfers.

When an instruction writes or reads a CP memory address matching the breakpoint address, (for the current domain only, in Job mode), bit 47 of the data flag register is set, indicating that a condition that can cause automatic branching has occurred. The data flag register can be set to cause a branch to a special routine provided for support of debugging operations, for example, a routine to trap the current program address.

Figure 1-1 shows the contents of register R for the breakpoint instruction.

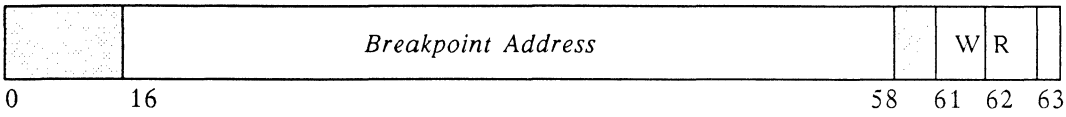


Figure 1-1. Structure of Register R for the #04 Instruction.

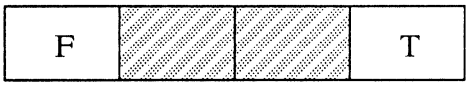
Bits 16-58 of register R hold the breakpoint address, the CP memory address which initiates the breakpoint function. Bits 61 and 62 can be set to specify the breakpoint function. Usage bit 61 specifies the breakpoint function for CP memory write instructions, and usage bit 62 specifies the breakpoint function for CP memory read instructions. Either or both bits 61 and 62 may be set. Bits 0-15, 59-60, and 63 are unused and must be cleared to zero.

In Job mode, the breakpoint address is saved in the breakpoint register, and stored in the current invisible package for mode or domain changes. Since a Job to Monitor mode change clears the breakpoint register, and Monitor mode has no invisible package saved, the monitor program must reload the breakpoint register if the breakpoint function is needed.

05

Void Stack and Branch

Half Word, Format #4
Subfunction: None



The #05 instruction voids the instruction stack, and branches to the address contained in register T.

Note: An #05 instruction should follow immediately after an instruction that stores modified code. This ensures that the code executed is the modified code.

06

Fault Test

Half Word, Format #9
Subfunction: None

The #06 instruction is used to complement the checkword bits on the Scalar Write bus so that the Read SECDED circuitry may be checked out. It is also used to disable the error correction circuitry on all read buses; this permits data to pass through the SECDED hardware without correction.

The #06 instruction's function is determined by bits set in the R designator. R-bit 0 is set to disable error correction on all Read buses, and R-bits 1-7 set to complement the seven checkword bits accompanying each 32-bit operand. When testing completes, the effect must be reversed by executing the instruction with the R designator cleared to zero.

These bits must be cleared to zero with the #06 instruction before any Monitor to Job Exchange Operation. If they are not cleared, the correction network could produce invalid data on the Read, and write invalid data into memory.

07

Select Serial/Parallel Execution Mode

Half Word, Format #7
Subfunction: None

The #07 instruction uses the R designator bit 7 to select the execution mode for CPU instructions that follow #07's execution. There are two instruction execution modes; serial, selected by setting R-bit 7 to one, and parallel, selected by setting R-bit 7 to zero.

In serial mode, no overlap or parallel operation of separate parts of different instructions occurs. Each instruction voids the instruction stack, is reread from memory, and completes with results properly stored, before the next instruction begins execution. A single instruction's execution time is unaffected by the choice of serial mode.

Parallel execution mode allows all overlap and parallel operations of separate parts of different instructions to the full extent of the machine capability. This is the normal mode after Master Clear, unless the CPU is in Force Execution mode. In this case, the instruction executes as a no-op.

Force Execution mode is selected or unselected by the service unit (SU). There are two bits in the maintenance unit input register of the CPU, set by the SU, which force the CPU to ignore the #07 instruction and allow the SU to select serial or parallel mode.

The execution mode remains in effect until a #07 instruction is executed with the other mode selected. The execution mode is unaffected by exchanges and domain changes. The #07 instruction can be executed in either Job or Monitor mode. Bit 03 of the Domain Package Illegal Instruction Mask must be cleared for execution in Job mode.

08

Transmit External Interrupt

Half Word, Format #4
Subfunction: None

The #08 instruction transmits an external interrupt to destinations selected by control bits in register R. Control bits are assigned to selected Central Processing Units (CPU), Input-Output Units (IOU), and the Service Unit (SU). The instruction is legal in both Job and Monitor mode. Bit 4 of the Domain Package Illegal Instruction Mask must be cleared for execution in Job mode.

Register R's structure is shown in figure 1-2. Bits 08-15, 26-31, and 42-62 are unassigned and must be cleared to zero.

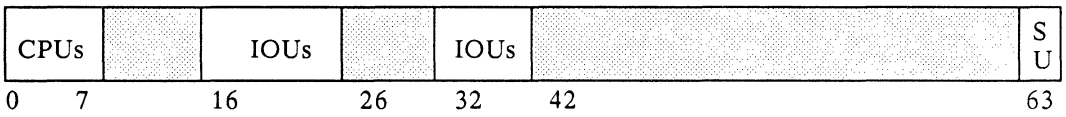


Figure 1-2. Structure of Register R for the #08 Instruction.

- Bits 0-7 are assigned to up to eight CPUs, numbered from 0 to 7. Interrupts are transmitted to CPUs 0 to 7 by setting bits from 0 to 7 in register R. The actual numbering of CPUs in a system is not necessarily sequential, and does not necessarily begin with 0. Only bits corresponding to CPUs configured in the system are assigned.

- Bits 16-25 and bits 32-41 are assigned to up to 18 IOUs, numbered from 0 to 8 and 10 to 18. Interrupts are transmitted to IOUs by setting bits in the assigned ranges of the R register. Bit 16 selects the service unit acting as an IOU, bit 17 selects IOU-0, up to bit 25, which selects IOU-08. Bit 32 selects the service unit acting as an IOU, bit 33 selects IOU-10, up to bit 41, which selects IOU-18. The numbering of IOU's is not necessarily sequential, and does not necessarily begin with zero. Only bits corresponding to IOUs configured in the system are assigned.

- Bit 63 is assigned to the Service Unit.

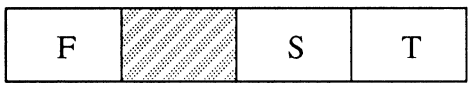
09

Exit Force

Half Word, Format #4
Subfunction: None

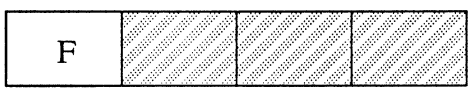
The #09 instruction transfers control from Monitor mode to Job mode, and from Job mode to Monitor mode. This transfer is called an exchange.

Exchange from Monitor Mode to Job Mode



The Monitor mode register file is stored at address zero in CP memory while the Job register file is loaded from the Job Register File package, and the process status registers are loaded from the Job invisible package. Execution of Job mode instructions begins at the program address in the invisible package. Register T contains the Job invisible package base address, an absolute bit address aligned on a 64-word boundary. Register S contains the exchange's job register file base address, an absolute bit address aligned on a 64-word boundary. If designator S is zero, or if the contents of Monitor's register S are absolute address zero, the Job Register File is the Monitor's Register File. The #09 instruction is undefined if S's contents are between zero and #4000, if there is overlap of CP memory space for the invisible Package or the Job Register File, or if there is overlap of the job's virtual storage space in CP memory, the job's invisible package, and Monitor's register File package.

Exchange from Job Mode to Monitor Mode



This instruction sets bit 62 of the interrupt register to cause an interrupt, and thus the exchange. The exchange from Job to Monitor mode is performed as for any other interrupt. The exchange stores the job register file and the invisible package at the addresses provided by the Monitor mode to Job mode exchange #09, and loads the Monitor register file. Execution of Monitor mode instructions begins at the absolute bit address in Monitor's register #03.

0A

Transmit (R) to Monitor Interval Timer

Half Word, Format #4
Subfunction: None

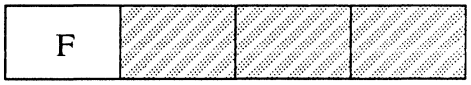
The #0A instruction is valid only in Monitor mode. It activates the Monitor interval timer by loading it with a non-zero value from bits 32-63 of register R. The left-most 32 bits of register R are ignored.

Once activated, the timer decrements at a 1-MHz rate until reaching zero, unless it is first deactivated. When the timer decrements to zero, it causes an interrupt by setting bit 60 of the interrupt register. The timer may be deactivated before reaching zero by reloading it with all 32 bits cleared, or by a master clear.

0C

Store Associative Registers

Half Word, Format #4
Subfunction: None

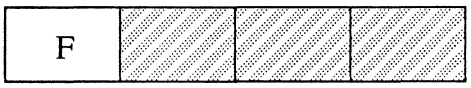


This instruction must be executed to update the first 16 entries in the page table. In Monitor mode, the contents of the associative registers are stored into absolute address #4000 and forward. The contents of the associative registers are undefined after the #0C instruction executes. Two #0C instructions without a #0D instruction between are undefined.

0D

Load Associative Registers

Half Word, Format #4
Subfunction: None



In Monitor mode, the contents of the associative registers are loaded from absolute address #4000 and forward.

0E

Read Interrupt Register to (T)

Half Word, Format #4
Subfunction: None

The #0E instruction executes in Monitor mode only. It moves the contents of the interrupt register (IR) into register T, and clears the interrupt register.

When the CPU receives an interrupt, an assigned bit in the interrupt register, representing the source of the interrupt, is set. Assigned bits remain set until an #0E instruction is executed. The interrupt register is cleared as it is read. Figure 1-3 shows the contents of the interrupt register.

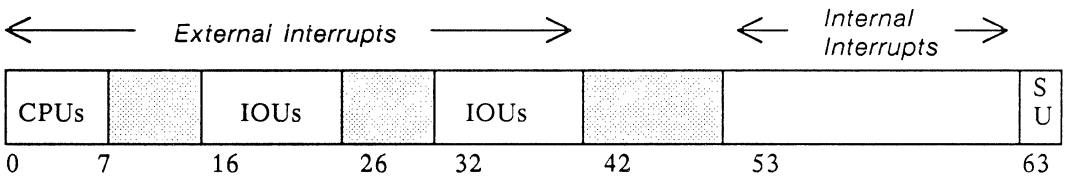


Figure 1-3. Register T after an #0E instruction.

- Bits 0-41 represent external interrupts sent from sources outside the CPU executing the instruction. These bit assignments reflect the system configuration. Bits 08-15, 26-31, and 42-52 are always unassigned and unused, and are always zero.
- Bits 53-62 represent internal interrupts sent from sources associated only with the CPU executing the instruction.
- Bit 63 is the destination for an interrupt sent from the Service Unit.

Unassigned bits of the interrupt register are always zeros. All assigned bits are cleared during a Master Clear.

External Interrupt Bit Assignments

- Bits 0–7, representing external interrupts, are assigned to up to eight CPUs, numbered from 0 to 7. An interrupt received from a CPU is recognized by setting the corresponding bit from 0 to 7 in the interrupt register. CPU–0 is recognized as an interrupt source by setting bit 0, CPU–1 is recognized by setting bit 1, and so on. CPU numbering is not necessarily sequential, and does not necessarily begin with 0. Only bits corresponding to CPUs configured in the system are assigned.

- Bits 16–41 represent external IOU interrupts. The system can be configured with up to 18 IOUs, numbered from 0 to 8 and 10 to 18; two bits in this range, 16 and 32, are reserved for the service unit acting as an IOU. An interrupt received from an IOU is recognized by setting the corresponding bit in the interrupt register. Bit 17 recognizes IOU–0 as an interrupt source, bit 18 recognizes IOU–1, up to bit 25, which recognizes IOU–8. Bit 33 recognizes IOU–10 as an interrupt source, bit 34 recognizes IOU–11, up to bit 41, which recognizes IOU–18. The numbering of IOU's is not necessarily sequential, and does not necessarily begin with zero. Only bits corresponding to IOUs configured in the system are assigned.

Internal Interrupt Bit Assignments

- Bit 53 is set by a shared memory hardware failure.

- Bit 54 is set by completion of a shared memory transfer request block (TRB).

- Bit 55 is set by an #FA–#FF instruction that is locked out of the Communication Buffer by an Access Lockout Code.

- Bit 56 is set by a #FA–#FF instruction that cannot access the communication buffer because of a base/limit addressing error.

- Bit 57 is set by a communication buffer hardware failure.

- Bit 58 is set by the execution of a type one illegal instruction.

- Bit 59 is set by the execution of a type two illegal instruction.

- Bit 60 is set when the Monitor interval timer decrements to zero.

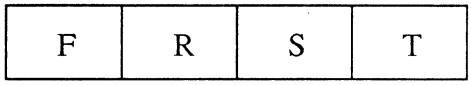
- Bit 61 is set by an access interrupt.

- Bit 62 is set by the #09 instruction executed in Job mode.

0F

Load Keys from (R), Translate Address (S) to (T)

Half Word, Format #4
Subfunction: None



The #0F instruction is executed in Monitor mode only. Register R contains four keys that are loaded into the virtual address key registers. The virtual address in the right-most 48 bits of register S is 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. The left-most 16 bits of register S are transmitted to the corresponding position in register T.

If no address translation is possible before reaching the end of the Page Table, the right-most 48 bits of register T are cleared. The Associative Word used to make the translation is left in the top Associative Register (register #00).

The Page Table is dynamically pushed down if necessary when searching for the Associative Word used to make the translation. The instruction uses the Page Table as contained in the Associative Registers and the Space Table in memory.

If the Associative Registers were not loaded by a #0D instruction, the operation is undefined. The 3-bit size, alteration and reference code in the associative word is not changed by this instruction. Register R's contents are described in figure 1-4.

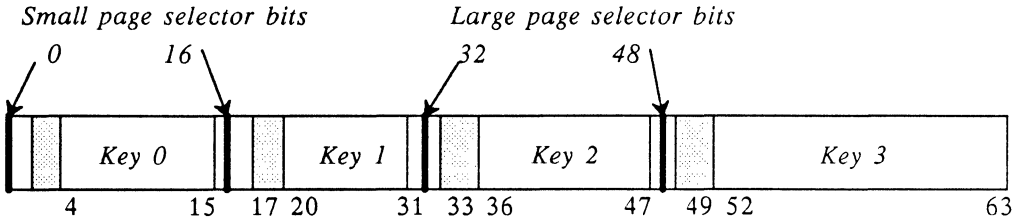


Figure 1-4. Structure of Register R for the #0F instruction.

- Bits 0 and 16 select the Job mode small page size. A small page size of 1K, 2K, or 8K is selected by setting bits 0 and 16 to 00, 10, or 11. The 01 combination is undefined.
- Bits 32 and 48 select the Job mode large page size. A large page size of 64K or 256K is selected by setting bits 32 and 48 to 00 or 01; the 10 and 11 combinations are undefined and not allowed.
- Bit 32 must be zero.
- Bits 1-3, 17-19, 33-35, and 49-51 are not used, and must be zero.

10

Convert BCD to Binary, Fixed Length

Half Word, Format #A
Subfunction: None

The #10 instruction converts the Binary Coded Decimal (BCD) number in register R to a signed two's complement binary number and places the result into the right-most 48 bits of register T. Bits 0–15 of register T are cleared to zero.

BCD representation can accommodate a signed 15-digit integer in one 64-bit word. The word is treated as sixteen 4-bit fields, with the right-most field (bits 60–63) used for the sign code. The fifteen remaining fields each contain one hexadecimal digit with a decimal value of nine or less. A BCD number is invalid if it has hexadecimal digits with decimal values of ten or larger in any of these fields. If the input value is not a valid BCD number, the results are undefined.

The sign code field must contain a hexadecimal digit with a decimal value of ten or larger. The sign of the BCD number is positive when the sign code is an even digit, or #F; the sign is negative when the sign code is an odd digit, except for #F.

The conversion is undefined for binary results greater than $(+2^{47}-1)$ or less than (-2^{47}). The largest decimal number that may be converted is ±140,737,488,355,327.

Data Flag Bit Settings:

Data Flag Bit 39: Input number is outside range.

11

Convert Binary to BCD, Fixed Length

Half Word, Format #A
Subfunction: None

The #11 instruction converts the right-most 48 bits of register R, interpreted as a two's complement binary number, to a Binary Coded Decimal (BCD) number, and places the result into 64-bit register T.

BCD representation can accommodate a signed 15-digit integer in one 64-bit word. The word is treated as sixteen 4-bit fields, with the right-most field (bits 60–63) used for the sign code. The fifteen remaining fields each contain one decimal digit with a value of nine or less.

The sign code field must contain a hexadecimal digit with a decimal value of ten or larger. The sign of the BCD number is positive when the sign code is an even digit, or #F; the sign is negative when the sign code is an odd digit, except for #F.

In Job mode, the sign code generated is determined by the ASCII/EBCDIC bit in the Job Invisible Package. ASCII sign codes are #A and #B for plus and minus; corresponding EBCDIC sign codes are #C and #D. In Monitor mode, only ASCII codes are generated.

12

Load Byte from CP Memory

Half Word, Format #7
Subfunction: None

F	R	S	T

(T) per (S), (R)

The #12 instruction loads a byte from the CP memory address specified by the sum of registers R and S, where R is the base address and S is an item count in bytes. The item count is shifted left three places before being added to the address in R.

The object byte is loaded into bits 56–63 of register T. The other bits of register T are cleared.

13

Store Byte to CP Memory

Half Word, Format #7
Subfunction: None

F	R	S	T

(T) per (S), (R)

The #13 instruction stores a byte into the CP memory address specified by the sum of registers R and S, where R is the base address and S is an item count in bytes. The item count is shifted left three places before being added to the address in R.

The object byte is taken from bits 56–63 of register T and put in CP memory. The other bits of register T are ignored.

14

Bit Compress

Half Word, Format #7
Subfunction: None

F	R	S	T

The #14 instruction compresses the bit field R, as specified by length S, into bit field T. The operation is performed from left to right. The left-most 16 bits of register R specify the number of bits to transfer at one time as a segment. Field R's base address is in the right-most 48 bits of register R.

The left-most 16 bits of register S specify the number of bits to skip in the R field between transferred bit segments. The remaining bits of register S are unused.

Register T contains the length and base address of the destination field. The left-most 16 bits are the field length; the destination's base address is in the right-most 48 bits. The destination's length need not be an integer multiple of the segment length. The field is filled with whatever portion of the last segment is needed.

The operation moves the left-most segment of R-field bits to the destination, then skips a number of bits in the R field equal to the S length. The next R segment is moved, S length bits skipped, and the pattern repeated until the destination is filled, as shown in figure 1-5.

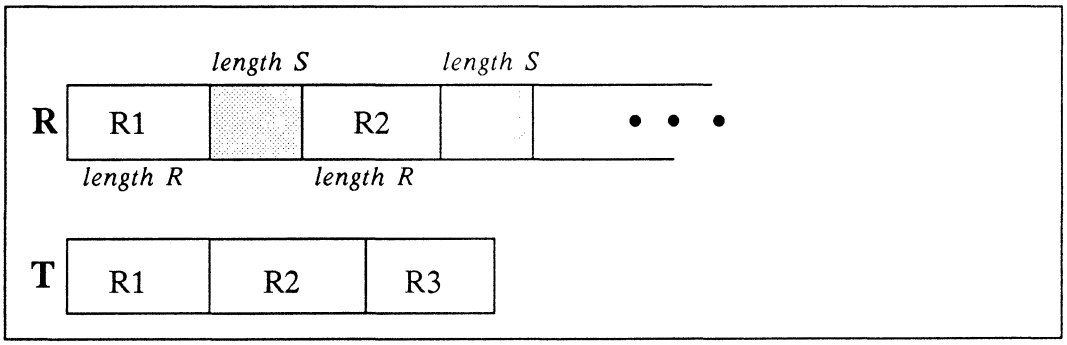


Figure 1-5. Bit Compress Operations.

The instruction is treated as a no-op if a zero field length is specified for source R or destination T.

15

Bit Merge

Half Word, Format #7
Subfunction: None

F	R	S	T

The Bit Merge instruction merges the bit fields R and S into the destination field T. The left-most bits (equal to the R segment length) of the R field, followed by the left-most bits (equal to the S segment length) of the S field, are moved to the left-most R & S bits of the destination field. These are followed by the next bit segments from R and S, repeating the pattern until the destination field is filled, shown in figure 1-6.

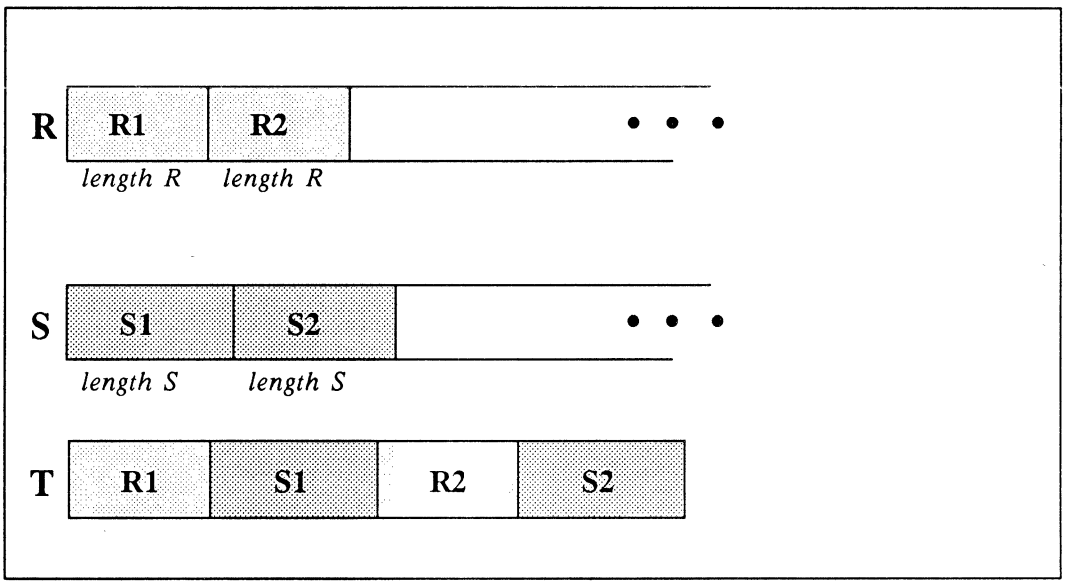


Figure 1-6. Bit Merge Operations.

The T field's length need not be an integer multiple of any segment length. The destination field is filled with whatever portion of the last segment is needed.

The left-most 16 bits of register R specify the number of bits to transfer from R at one time as a segment. The base address is in the right-most 48 bits of register R.

The left-most 16 bits of register S specify the number of bits to transfer from S at one time as a segment. The right-most 48 bits of register S contain the base address. If the S base address is zero, a zero filled S field is used.

Register T contains the destination's length and base address. The left-most 16 bits is the field length; the base address is in the right-most 48 bits.

The instruction is treated as a no-op if a zero field length is specified for R, S, or T.

16

Bit Mask

Half Word, Format #7
Subfunction: None

F	R	S	T

The #16 instruction masks the bit fields R and S into field T, working from left to right. The operation moves a segment of bits (bits 0–15 of the R register specify the segment length) from the R field to the T field. Next it moves to T a segment of bits (bits 0–15 of the S register specify the segment length) from the S field, starting at the S base address plus the R field length. The next segment is moved to T from the R field, starting at the R base address plus the R and S segment lengths.

This pattern of selecting bits equal to the R segment length and skipping bits equal to the S segment length in the R source field, then selecting S-length bits and skipping R-length bits in the S source field, is repeated until the destination field is filled, as shown in figure 1–7. The shaded areas are not moved to T.

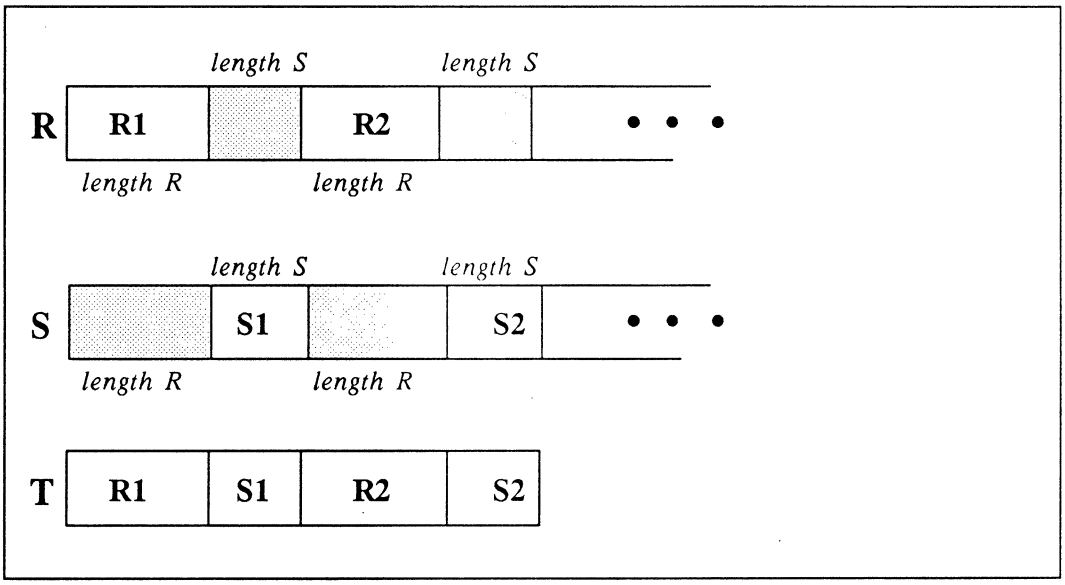


Figure 1–7. Bit Mask Operations.

The T field's length need not be an integer multiple of any segment length. The field is filled with whatever portion of the last segment is needed.

The left-most 16 bits of register R specify the number of bits of the R field to move (and the S field to skip); the base address of the R field is in the right-most 48 bits. The left-most 16 bits of register S specify the number of the S field bits to move (and R field bits to skip); the right-most 48 bits contain the S field's base address. If the base address in bits 16–63 of register S is zero, a zero filled S source field is used.

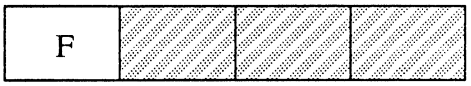
Register T contains the destination's length in the left-most 16 bits, and the base address in the right-most 48 bits.

The 16 instruction is treated as a no-op if the field length specified for R, S, or T is zero.

17

Backward Domain Change

Half Word, Format #7
Subfunction: None

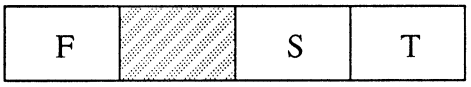


The #17 instruction is defined only in Job mode. It is the last instruction coded for execution in a domain type subroutine. It returns control to the calling program in the domain specified by the stacked domain package at the top of the stacked domain package stack. The number of backward domain changes must not exceed the number of forward domain changes for a program. A forward domain change must always precede the corresponding backward domain change. Refer to the #36 instruction description and PUB-1005, *ETA10 System Reference Manual.*

18

Shared Memory Transfer

Half Word, Format #7
Subfunction: None



CQTA to (T), (S) to CQTA

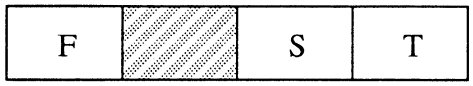
The #18 instruction clears register T and transfers the contents of the Completion Queue Tail Address (CQTA) register to bits 32-63 of register T. Bits 32-63 of register S are then transferred to the CQTA register.

If register S is the same as register T, a swap operation occurs between bits 32-63 of register S or T and the CQTA register.

19

Shared Memory: Start I/O

Half Word, Format #7
Subfunction: None



(S) to IQHA, (T) TO IQTA, Start Transfer

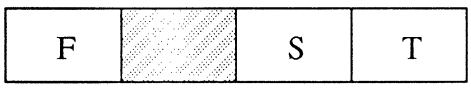
The #19 instruction transfers bits 32–63 of register S to the Input Queue Head Address (IQHA) register, and transfers bits 32–63 of register T to the Input Queue Tail Address (IQTA) register. It also sets the Input Queue Valid Flag (IQVF).

If S and T specify the same register, or if bits 32–57 of register S equal bits 32–57 of register T, one Transfer Request Block (TRB) will be executed. Bits 58–63 of registers S and T are ignored for the address compare operation.

1A

Shared Memory: Stop I/O

Half Word, Format #7
Subfunction: None



IQHA to (S), IQVF and IQTA to (T)

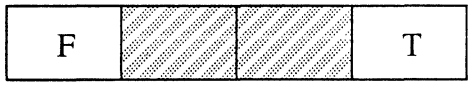
The #1A instruction clears registers S and T, then transfers the contents of the Input Queue Head Address (IQHA) register to bits 32–63 of register S. Next, it transfers the Input Queue Valid Flag (IQVF) to bit 0, and the contents of the Input Queue Tail Address (IQTA) register to bits 32–63 of register T. The Input Queue Valid Flag is then cleared.

Results are undefined if S and T specify the same register.

1B

Shared Memory: Test I/O

Half Word, Format #7
Subfunction: None



IQVF, Transfer Busy Flag, Fatal Error Status and TRBSA to T

The #1B instruction clears register T, then transfers the Input Queue Valid Flag (IQVF) to bit 0, the "transfer busy" flag to bit 1, the fatal error status to bits 2–9, and the contents of the Transfer Request Block Store Address (TRBSA) register to bits 32–63 of register T.

If the "transfer busy" flag is clear and the "termination with fatal error" bit is set, the IQVF bit and all fatal error status bits are cleared. The fatal error status bits are not valid until the transfer busy flag has dropped from one to zero. The fatal error status bits are:

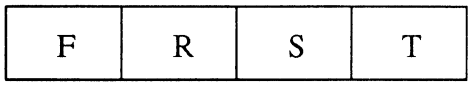
- Bit 2: Termination with fatal error.
- Bit 3: CP memory to shared memory address parity error.
- Bit 4: CP memory to shared memory data parity error.
- Bit 5: Shared memory double SECDED error.
- Bit 6: Shared memory boundary error. Bit 6 is set if any single transfer attempts to reference both halves of shared memory.
- Bit 7: Shared memory to CP memory data parity error.
- Bit 8: CP Memory double SECDED error.
- Bit 9: CP Memory double SECDED error occurred while fetching this TRB. Bit 9, if set, will block all write enables to Shared memory or CP memory during the data transfer and during the store TRB operation. It will also block any updating of the CQTA register.

Bit 2 is set for any fatal error, and cleared if there is no fatal error. Bits 3–9 are set for a fatal error, and cleared for no fatal error.

1C

Form Repeated Bit Mask with Leading Zeros

Half Word, Format #7
Subfunction: None



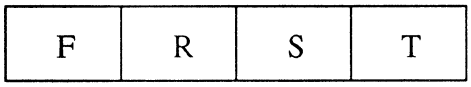
The #1C instruction forms a repeated mask in field T, consisting of a string of zeros followed by a string of ones. The left-most 16 bits of register R specify the length in bits of the string of zeros. The left-most 16 bits of register S specify the length in bits of the repeated mask (the string of zeros plus the string of ones). Field T's length in bits and starting address are located in the left-most 16 bits and the right-most 48 bits of register T, respectively.

If length R exceeds length S, the instruction is undefined. If the lengths are the same, a string of zeros is formed. If length R is zero, a string of ones is formed. If length S is zero, the instruction performs as a no-op. The instruction terminates when the T field is filled.

1D

Form Repeated Bit Mask with Leading Ones

Half Word, Format #7
Subfunction: None



The #1D instruction forms a repeated mask in field T, consisting of a string of ones followed by a string of zeros. The left-most 16 bits of register R specify the length in bits of the string of ones. The left-most 16 bits of register S specify the length in bits of the repeated mask (the string of ones plus the string of zeros). Field T's length in bits and starting address are located in the left-most 16 bits and the right-most 48 bits of register T, respectively. The instruction terminates when the T field is filled.

If length R exceeds length S, the instruction is undefined. If the lengths are the same, a string of ones is formed. If length R is zero, a string of zeros is formed. If length S is zero, the instruction performs as a no-op.

1E

Count Leading Equals

Half Word, Format #7
Subfunction: None

F	R	S	T

The #1E instruction scans the bits in field R from left to right, until encountering a bit that is not equal to the left-most bit. The operation starts with the bit to the immediate right of the left-most bit of the field. The count of equal bits is stored in the right-most bits of register T. Register T is cleared before the count is stored. The left-most 16 bits of register R specify the length in bits of the field, and the right-most 48 bits specify the field's base address. Register S contains an index in bits that is added to the base address to form the R field's starting bit address.

The instruction terminates either when it encounters a bit unequal to the left-most field bit, or when the entire field has been scanned. In the latter case, the count stored is the field length minus one. Data Flag bit 53 is cleared when #1E is initiated, and set to one if the left-most bit was a one.

1F

Count Ones in Field R, Count to (T)

Half Word, Format #7
Subfunction: None

F	R	S	T

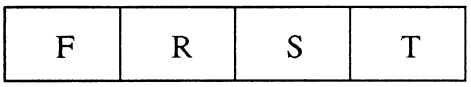
The #1F instruction scans bits in field R from left to right, counting the number of binary ones. The count is stored in the right-most bits of register T. Register T is cleared before the count is stored.

The left-most 16 bits of register R specify the length in bits of field R, and the right-most 48 bits hold the field's base address. Register S contains an index in bits that is added to the base address to form the R field's starting bit address. The instruction terminates when the entire field has been scanned.

20

Branch if Equal (32-Bit)

Half Word, Format #8
Subfunction: None



(R) EQ (S)

The #20 instruction compares the 32-bit floating-point operands in registers R and S, then conditionally branches to the address in register T. Refer to floating point comparison rules in Appendix F.

The S operand is subtracted from the R operand, and compared according to floating-point comparison rules. If the operands are equal, the next instruction is read from the address in register T. If the comparison fails, the next instruction is read from the next sequential program address.

Data flag branch conditions:

Data flag bit 46: Set if either or both operands are indefinite.

21

Branch if Not Equal

Half Word, Format #8
Subfunction: None

F	R	S	T

(R) NE (S) (32-Bit FP)

The #21 instruction compares the 32-bit floating-point operands in registers R and S, then conditionally branches to the address in register T. Refer to floating point comparison rules in Appendix F.

The S operand is subtracted from the R operand, and compared according to floating-point comparison rules. If the operands are equal, the next instruction is read from the address in register T. If the comparison fails, the next instruction is read from the next sequential program address.

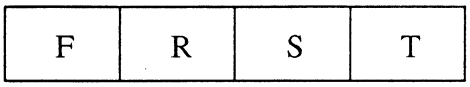
Data flag branch conditions:

Data flag bit 46: Set if either or both operands are indefinite.

22

Branch if Greater or Equal (32-Bit FP)

Half Word, Format #8
Subfunction: None



(R) GE (S)

The #22 instruction compares the 32-bit floating-point operands in registers R and S, then conditionally branches to the address in register T. Refer to floating point comparison rules in Appendix F.

The S operand is subtracted from the R operand, and compared according to floating-point comparison rules. If R is greater than or equal to S, the next instruction is read from the address in register T. If the comparison fails, the next instruction is read from the next sequential program address.

Data flag branch conditions:

Data flag bit 46: Set if either or both operands are indefinite.

23

Branch if Less (32-Bit FP)

Half Word, Format #8
Subfunction: None

F	R	S	T

(R) LT (S)

The #23 instruction compares the 32-bit floating-point operands in registers R and S, then conditionally branches to the address in register T. Refer to floating point comparison rules in Appendix F.

The S operand is subtracted from the R operand, and compared according to floating-point comparison rules. If R is less than S, the next instruction is read from the address in register T. If the comparison fails, the next instruction is read from the next sequential program address.

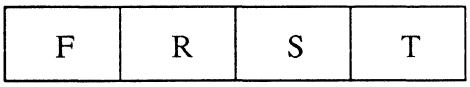
Data flag branch conditions:

Data flag bit 46: Set if either or both operands are indefinite.

24

Branch if Equal (64-Bit FP)

Half Word, Format #8
Subfunction: None



(R) EQ (S)

The #24 instruction compares the 64-bit floating-point operands in registers R and S, then conditionally branches to the address in register T. Refer to floating point comparison rules in Appendix F.

The S operand is subtracted from the R operand, and compared according to floating-point comparison rules. If the operands are equal, the next instruction is read from the address in register T. If the comparison fails, the next instruction is read from the next sequential program address.

Data flag branch conditions:

Data flag bit 46: Set if either or both operands are indefinite.

25

Branch if Not Equal (64-Bit FP)

Half Word, Format #8
Subfunction: None

F	R	S	T

(R) NE (S)

The #25 instruction compares the 64-bit floating-point operands in registers R and S, then conditionally branches to the address in register T. Refer to floating point comparison rules in Appendix F.

The S operand is subtracted from the R operand, and compared according to floating-point comparison rules. If the operands are not equal, the next instruction is read from the address in register T. If the comparison fails, the next instruction is read from the next sequential program address.

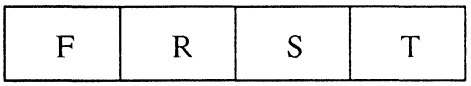
Data flag branch conditions:

Data flag bit 46: Set if either or both operands are indefinite.

26

Branch if Greater or Equal (64-Bit FP)

Half Word, Format #8
Subfunction: None



(R) GE (S)

The #26 instruction compares the 64-bit floating-point operands in registers R and S, then conditionally branches to the address in register T. Refer to floating point comparison rules in Appendix F.

The S operand is subtracted from the R operand, and compared according to floating-point comparison rules. If R is greater than or equal to S, the next instruction is read from the address in register T. If the comparison fails, the next instruction is read from the next sequential program address.

Data flag branch conditions:

Data flag bit 46: Set if either or both operands are indefinite.

27

Branch if Less (64-Bit FP)

Half Word, Format #8
Subfunction: None

F	R	S	T

(R) LT (S)

The #27 instruction compares the 64-bit floating-point operands in registers R and S, then conditionally branches to the address in register T. Refer to floating point comparison rules in Appendix F.

The S operand is subtracted from the R operand, and compared according to floating-point comparison rules. If R is less than S, the next instruction is read from the address in register T. If the comparison fails, the next instruction is read from the next sequential program address.

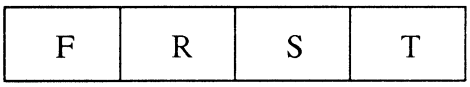
Data flag branch conditions:

Data flag bit 46: Set if either or both operands are indefinite.

28

Scan for Equal Byte

Half Word, Format #7
Subfunction: None



The #28 instruction scans the bytes in field T, indexed by S, from left to right, looking for the first byte equal to byte R.

The right-most 48 bits of register S contains an index, which is an item count in bytes, shifted left three places before being added to T's base address. The scan stops at the first byte in the T field that equals byte R (designator R). The index is incremented by the number of bytes scanned before the byte was found. If no equal byte is found, the index is incremented by the number of bytes in the T field. The updated index is then written into register S.

The left-most 16 bits of register T contain the field's length in bytes, and the right-most 48 bits contain the field's base address.

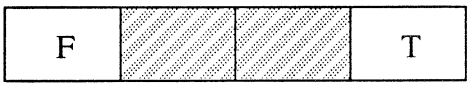
Data flag branch conditions:

Data flag bit 53: Set if no equal byte is found.

29

Transmit Instrumentation Counter to (T)

Half Word, Format #A
Subfunction: None

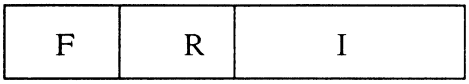


The #29 instruction transmits the contents of Instrumentation Counter 7 (a CPU cycle counter) to the right-most 48 bits of register T. Bits 0–15 of register T are cleared.

2A

Enter Length of (R) with I (16 Bits)

Half Word, Format #6
Subfunction: None

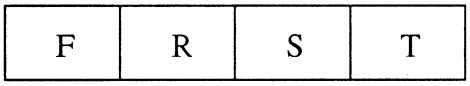


The #2A instruction transfers the 16-bit immediate operand (I) to the left-most 16 bits of register R. The right-most 48 bits of register R are unchanged.

2B

Add to Length Field

Half Word, Format #4
Subfunction: None



The #2B instruction adds bits 0–15 of the 64-bit register R to bits 48–63 of the 64-bit register S. The result is stored in bits 0–15 of register T. Bits 16–63 of register R are moved to bits 16–63 of register T.

2C

Logical Exclusive OR

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) Excl. OR (S) To (T)

The #2C instruction performs a bit-by-bit logical exclusive OR operation on the 64-bit operands in registers R and S. The result is stored in register T. If designator R or S is zero, register #00 provides machine zero for the operation. The results, based on bit settings in the R and S registers, are:

R	S	Excl. OR
0	0	0
0	1	1
1	0	1
1	1	0

2D

Logical AND

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) AND (S) To (T)

The #2D instruction performs a bit-by-bit logical AND operation on the 64-bit operands in registers R and S. The result is stored in register T. If designator R or S is zero, register #00 provides machine zero for the operation. The results, based on bit settings in the R and S registers, are:

R	S	AND
0	0	0
0	1	0
1	0	0
1	1	1

2E

Logical Inclusive OR

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) OR (S) To (T)

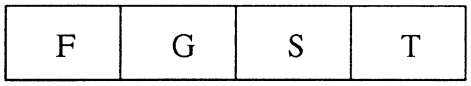
The #2E instruction performs a bit-by-bit logical inclusive OR operation on the 64-bit operands in registers R and S. The result is stored in register T. If designator R or S is zero, register #00 provides machine zero for the operation. The results, based on bit settings in the R and S registers, are:

R	S	Incl. OR
0	0	0
0	1	1
1	0	1
1	1	1

2F

Register Bit Branch and Alter

Half Word, Format #9
Subfunction: bboo0aa0
Qualifiers: bb =[br,bro,brz],oo=[t,so,sz],aa=[brb,brf]



The #2F instruction examines bit 63 (the object bit) in register T, and, depending on the specified branch *(bb)* and bit modification *(oo)* qualifiers, branches to the address in the right-most 48 bits of register S, if *aa* is not specified. The operation may also change the value of the object bit.

If no *bb* qualifier is specified, then G-bits 0 and 1 are clear and there is no branch. If the qualifier is *br* (G-bit 1), an unconditional branch occurs. The *bro* qualifier (G-bit 0) causes a branch if the object bit is one. If the qualifier is *brz* (G-bits 0 and 1), a branch occurs if the object bit is zero.

After the branch decision is made, the object bit is altered if an *oo* qualifier (G-bits 2 and 3) is specified. The *t* qualifier (G-bit 3) toggles the object bit's state. The object bit is set to one if the qualifier is *so* (G-bit 2), and cleared to zero if the qualifier is *sz* (G-bits 2 and 3).

If a branch is to take place, the instruction determines the branch address depending on the specified *aa* qualifier (G-bits 5 and 6). If no qualifier is specified, the address in register S is branched to.

The *brf* and *brb* qualifiers indicate that a relative branch will be taken to an address formed from a half word item count in the S designator and the program address register. The type of relative branch (forward or backward) depends on the specified qualifier.

If *brf* is specified, a forward branch occurs to the address formed by shifting the item count in register S left 5 places, and adding it to the program address register. *brb* specifies a backward branch to the address formed by shifting the item count in register S left 5 places and subtracting it from the program address register.

30

Shift Operand

Half Word, Format #7
Subfunction: None

F	R	S	T

(R) per S to (T)

The #30 instruction shifts the 64-bit operand in register R, and stores the result in the destination register T. Designator S specifies the type and amount of the shift. If the shift count is between #0 and #3F, the operand in register R is shifted left end-around for the number of specified places before being stored. If the shift count is between #FF and #C1, the operand in register R is shifted right, with sign extension. Bit zero of the operand is considered to be the sign bit of the shifted operand. The number of right shifts equals the two's complement of the S designator; for example, if the shift count is #FE, the operand is shifted right two places. If the shift count is greater than #3F or less than #C1, results are undefined. If the R designator is zero, register #00 provides a machine zero value.

31

Increase (R) and Branch

Half Word, Format #7
Subfunction: None

F	R	S	T

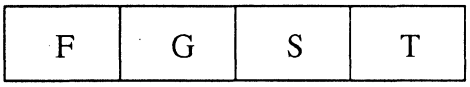
(R) NE Zero

The #31 instruction increments the right-most 48 bits of register R by one, and branches according to the result. The left-most 16 bits of register R are unchanged. Arithmetic overflow is ignored. If the result is 48 zeros, the next sequential instruction is executed. Otherwise, control branches to S + T, where register S contains an item count of half words, and register T contains the base address. If register R is the same as S or T, the resulting branch address is undefined.

32

Bit Branch and Alter

Half Word, Format #9
Subfunction: bboo0aa0
Qualifiers: bb=[br,bro,brz],oo=[t,so,sz],aa=[brb,brf]



The #32 instruction reads the word from memory from the address in register S and examines the object bit. Depending on the specified branch *(bb)* and bit modification *(oo)* qualifiers, it then branches to the address per the *aa* qualifier. The operation may also change the value of the object bit.

If no *bb* qualifier is specified, then G-bits 0 and 1 are clear and there is no branch. If the qualifier is *br* (G-bit 1), an unconditional branch occurs. The *bro* qualifier (G-bit 0) causes a branch if the object bit is one. If the qualifier is *brz* (G-bits 0 and 1), a branch occurs if the object bit is zero.

After the branch decision is made, the object bit is altered if an *oo* qualifier (G-bits 2 and 3) is specified. The *t* qualifier (G-bit 3) toggles the object bit's state. The object bit is set to one if the qualifier is *so* (G-bit 2), and cleared to zero if the qualifier is *sz* (G-bits 2 and 3).

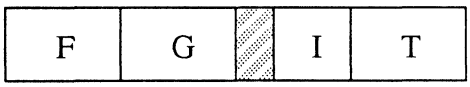
If a branch is to take place, the instruction determines the branch address depending on the specified *aa* qualifier (G-bits 5 and 6). If no qualifier is specified, the address in register T is branched to.

The *brf* and *brb* qualifiers indicate that a relative branch will be taken to an address formed from the T designator taken as a half word item count and the program address register. The type of relative branch (forward or backward) depends on the specified qualifier.

33

Data Flag Register Bit Branch and Alter

Half Word, Format #B
Subfunction: bboo0aa0
Qualifiers: b b=[br,bro,brz],oo=[t,so,sz],aa=[brb,brf]



The #33 instruction examines the object bit in the Data Flag Register specified by I, a 6-bit designator containing the number of a bit (between #00 and #3F). Depending on the specified branch *(bb)* and bit modification *(oo)* qualifiers, it then branches to the address per the *aa* qualifier. The operation may also change the value of the object bit in the Data Flag Register.

If no *bb* qualifier is specified, then G-bits 0 and 1 are clear and there is no branch. If the qualifier is *br* (G-bit 1), an unconditional branch occurs. The *bro* qualifier (G-bit 0) causes a branch if the object bit is one. If the qualifier is *brz* (G-bits 0 and 1), a branch occurs if the object bit is zero.

After the branch decision is made, the object bit is altered if an *oo* qualifier (G-bits 2 and 3) is specified. The *t* qualifier (G-bit 3) toggles the object bit's state. The object bit is set to one if the qualifier is *so* (G-bit 2), and cleared to zero if the qualifier is *sz* (G-bits 2 and 3).

If a branch is to take place, the instruction determines the branch address depending on the specified *aa* qualifier (G-bits 5 and 6). If no qualifier is specified, the address in register T is branched to.

The *brf* and *brb* qualifiers indicate that a relative branch will be taken to an address formed from the T designator, taken as a half word item count, and the program address register. The type of relative branch (forward or backward) depends on the specified qualifier.

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

Instructions that set Data Flag bits 53, 54, and 55 will always set these bits prior to execution of this instruction.

34

Shift Operand

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) per (S) to (T)

The #34 instruction shifts the 64-bit operand in register R according to a count in register S. The result is stored in the destination register T. If the shift count is between #0 and #3F, the operand in register R is shifted left end-around for the number of specified places before being stored.

If the shift count is between #FF and #C1, the operand in register R is shifted right, with sign extension. Bit zero of the operand is considered to be the sign bit of the shifted operand.

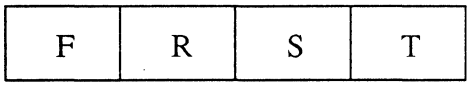
The number of right shifts equals the two's complement of the rightmost byte; for example, if the shift count is #FE, the operand is shifted right two places. If the shift count is greater than #3F or less than #C1, there are undefined results.

If the R designator is zero, register #00 provides a machine zero value.

35

Decrease (R) and Branch

Half Word, Format #7
Subfunction: None



(R) NE Zero

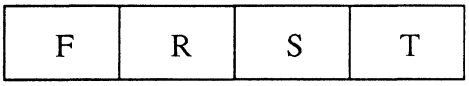
The #35 instruction decrements the right-most 48 bits of register R by one, and branches according to the result. The left-most 16 bits of register R are unchanged, and arithmetic overflow is ignored.

If the result is 48 zeros, the next sequential instruction is executed. Otherwise, a branch occurs to S + T, where register S contains an item count of half words, and register T contains the base address. If register R is the same as S or T, the resulting branch address is undefined.

36

Branch or Forward Domain Change

Half Word, Format #7
Subfunction: None



The #36 instruction performs one of two operations, a branch to a subroutine, or a forward domain change (in Job mode only). The operation performed depends on the R and T designators and bit 0 of register T.

Branch Operation

If bit 0 of register T is zero, or if designators R and T are equal, control branches to a subroutine within the current domain. The branch operation is undefined when registers R and S are the same, unless register #00 is designated.

The instruction stores the address of the next sequential instruction (the current program address *P*, plus 32) in register R, then branches to S + T, where register S contains an index of half words and register T contains the base address. The index is left-shifted 5 bits for use in computing the next instruction's address. Bits 0–15 of R are forced to zero, and bits 59–63 are undefined.

If the R and T designators are the same, a relative branch occurs to the address (S + *P* + 32), where register S contains an index of half words.

If register #00 is designated as S, or if register S is loaded with a zero value, the current program address, plus 32, is stored in register R, and execution continues with the next sequential instruction.

Forward Domain Change Operation

If bit 0 of register T is 1, and the R and T designators are not equal, a forward domain change occurs. (The loader generates a forward domain change (in Job mode only) when a process's permissions must be changed to execute a specified subroutine.)

Part of the invisible package information for the current domain is saved in the domain package and the stacked domain package. Control is then transferred to the next domain. When execution in the next domain is complete, the #17 backward domain change instruction is used to return control to the calling domain.

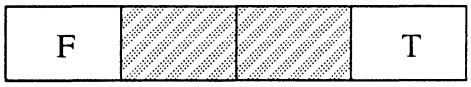
Each defined domain has its own domain package. When a forward domain change instruction executes, a stacked domain package is added to the stacked domain package stack for the current domain, and the instrumentation counters are saved in the current domain package. When the corresponding backward domain change instruction executes, the stacked domain package is loaded and deleted from the stack. The instrumentation counters from the domain package are also loaded.

The R and S designators are not defined for a forward domain change.

37

Transmit Job Interval Timer to (T)

Half Word, Format #A
Subfunction: None

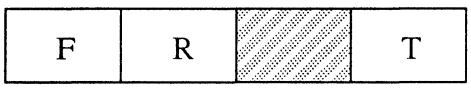


The #37 instruction transmits the contents of the Job Interval Timer into bits 32–63 of register T. Bits 0–31 of register T are cleared to zero. The timer is not deactivated. The instruction is undefined in Monitor mode.

38

Transmit (R) Bits 0–15 to (T) Bits 0–15

Half Word, Format #A
Subfunction: None

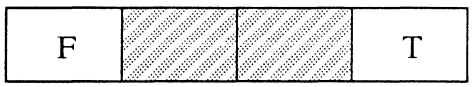


The #38 instruction replaces the left-most 16 bits of register T with the left-most 16 bits of register R.

39

Transmit Real Time Clock to (T)

Half Word, Format #A
Subfunction: None



The #39 instruction transmits the contents of the Real-Time Clock to bits 16 through 63 of register T. Bits 0 through 15 are cleared.

3A

Transmit (R) to Job Interval Timer

Half Word, Format #A
Subfunction: None

When executed in Job mode, this instruction transmits bits 32 through 63 of register R to the Job Interval Timer. In Monitor mode, the instruction performs as a no-op.

3B

Data Flag Register Load/Store

Half Word, Format #A
Subfunction: None

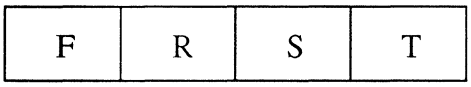
The #3B instruction transfers register R's contents to the Data Flag Register, and moves the original contents of the Data Flag Register to register T. The transfer to and from the Data Flag Register only occurs when all outstanding operations (except the job interval timer and breakpoint) affecting the data flags are complete. If a Data Flag Branch condition occurs during this time, no branch is taken, but the condition is stored in register T. If the R and T designators are the same, data flag packages will be swapped.

If the new Data Flag Register contents meet the appropriate conditions, a Data Flag Branch results.

3C

Half Word Index Multiply

Half Word, Format #4
Subfunction: None



(R) * (S) to (T)

The right-most 24 bits of registers 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 zero.

The result is undefined if the product exceeds $2^{23}-1$ or is less than -2^{23}.

3D

Index Multiply

Half Word, Format #4
Subfunction: None

F	R	S	T

(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 zero.

The result is undefined if the product exceeds $2^{47}-1$ or is less than -2^{47}.

3E

Enter (R) with I (16 Bits)

Half Word, Format #6
Subfunction: None

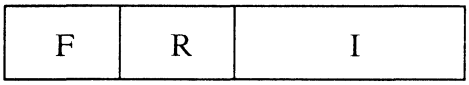
F	R	I

The #3E instruction clears register R and transfers the right-most 16 bits of this instruction (the immediate operand) to the right-most 48 bits of register R. The sign of the 16-bit immediate operand is extended through bit 16 of R.

3F

Increase (R) by I (16 Bits)

Half Word, Format #6
Subfunction: None

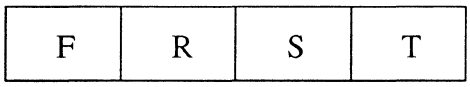


The #3F instruction replaces the right-most 48 bits of register R by the sum of those bits and the immediate operand (the right-most 16 bits of this instruction). The sign of the 16-bit immediate operand is extended through bit 16 for the addition. Arithmetic overflow is ignored.

40

Add; Upper Result (32 Bits)

Half Word, Format #4
Subfunction: None



(R) + (S) to (T)

The #40 instruction performs floating-point addition of the contents of the 32-bit registers R and S, returning the upper result in register T.

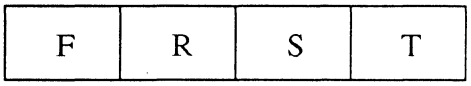
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero
Data flag bit 46:	Result is indefinite

41

Add; Lower Result (32 Bits)

Half Word, Format #4
Subfunction: None



(R) + (S) to (T)

The #41 instruction performs floating-point addition of the contents of the 32-bit registers R and S, returning the lower result in register T.

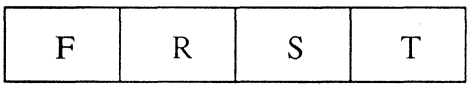
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero
Data flag bit 46:	Result is indefinite

42

Add; Normalized Result (32 Bits)

Half Word, Format #4
Subfunction: None



(R) + (S) to (T)

The #42 instruction performs floating-point addition of the contents of the 32-bit registers R and S, returning the normalized upper result in register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero
Data flag bit 46:	Result is indefinite

44

Subtract; Upper Result (32 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) – (S) to (T)

The #44 instruction performs floating-point subtraction of the contents of the 32-bit registers R and S, returning the upper result in register T.

Data flag branch conditions:

Data flag bit 42: Exponent overflow.
Data flag bit 43: Result is machine zero
Data flag bit 46: Result is indefinite

45

Subtract; Lower Result (32 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) – (S) to (T)

The #45 instruction performs floating-point subtraction of the contents of the 32-bit registers R and S, returning the lower result in register T.

Data flag branch conditions:

Data flag bit 42: Exponent overflow.
Data flag bit 43: Result is machine zero
Data flag bit 46: Result is indefinite

46

Subtract; Normalized Result (32 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) - (S) to (T)

The #46 instruction performs floating-point subtraction of the contents of the 32-bit registers R and S, returning the normalized upper result in register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero
Data flag bit 46:	Result is indefinite

48

Multiply; Upper Result (32 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) * (S) to (T)

The #48 instruction performs floating-point multiplication of the contents of the 32-bit registers R and S, returning the upper result in register T.

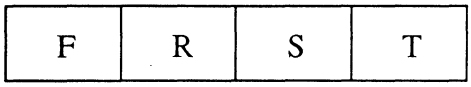
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero
Data flag bit 46:	Result is indefinite

49

Multiply; Lower Result (32 Bits)

Half Word, Format #4
Subfunction: None



(R) * (S) to (T)

The #49 instruction performs floating-point multiplication of the contents of the 32-bit registers R and S, returning the lower result in register T.

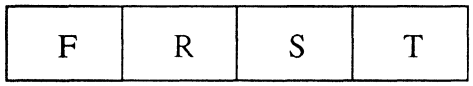
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero
Data flag bit 46:	Result is indefinite

4B

Multiply; Significant Result (32 Bits)

Half Word, Format #4
Subfunction: None



(R) * (S) to (T)

The #4B instruction performs floating-point multiplication of the contents of the 32-bit registers R and S, returning the significant result in register T.

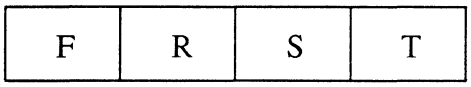
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero
Data flag bit 46:	Result is indefinite

4C

Divide; Upper Result (32 Bits)

Half Word, Format #4
Subfunction: None



(R) / (S) to (T)

The #4C instruction performs floating-point division of the contents of the 32-bit registers R and S, returning the upper result in register T.

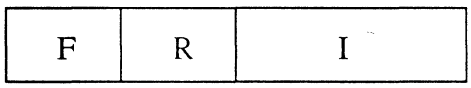
Data flag branch conditions:

Data flag bit 41:	Floating-point divide fault
Data flag bit 42:	Exponent overflow
Data flag bit 43:	Result is machine zero
Data flag bit 46:	Result is indefinite

4D

Half Word Enter R with I (16 Bits)

Half Word, Format #6
Subfunction: None

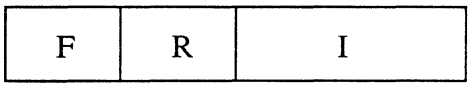


The #4D instruction clears register R and moves the 16-bit immediate operand I to the right-most 24 bits of 32-bit register R. The sign of the 16-bit operand is extended through bit 8 of R.

4E

Half Word Increase R by I (16 Bits)

Half Word, Format #6
Subfunction: None

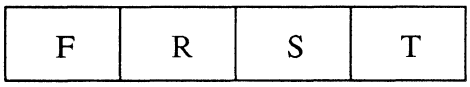


The #4E instruction adds the 16-bit immediate operand I to the right-most 24 bits of register R. I's sign is extended left through bit 8 before the addition. Arithmetic overflow is ignored.

4F

Divide; Significant Result (32 Bits)

Half Word, Format #4
Subfunction: None



(R) / (S) to (T)

The #4F instruction performs a floating-point divide significant operation on 32-bit register R's contents by the contents of 32-bit register S. The significant part of the floating-point result is stored in 32-bit register T.

Data flag branch conditions:

Data flag bit 41:	Floating-point divide fault.
Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Result is indefinite.

50

Truncate (32 Bits)

Half Word, Format #A
Subfunction: None

(R) to (T)

The #50 instruction transmits to 32-bit register T the nearest integer whose magnitude is less than or equal to the 32-bit floating-point operand in 32-bit register R. The integer is an unnormalized, 32-bit floating-point number with a positive exponent.

If R's exponent is positive, the operand is moved directly to T. If R's exponent is negative, the operation shifts the magnitude of the coefficient right, end-off, and increases the exponent by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, zeros are extended on the left, regardless of the sign bit value. For positive coefficients, the shifted coefficient with zero exponent is moved into 32-bit register T. For negative coefficients, the two's complement of the shifted coefficient, with zero exponent, is moved.

If machine zero is the operand value, 32 zeros are returned as the result.

Data flag branch conditions:

Data flag bit 46: Result is indefinite.

51

Floor (32 Bits)

Half Word, Format #A
Subfunction: None

(R) to (T)

The #51 instruction transmits to register T the nearest integer less than or equal to the 32-bit floating-point operand in 32-bit register R. The integer is an unnormalized, 32-bit floating-point number with a positive exponent.

If R's exponent is positive, the operand is moved directly to T. If R's exponent is negative, the operation shifts the coefficient right, end-off, and increases the exponent by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, sign bits are extended on the left. The shifted coefficient with zero exponent is moved into 32-bit register T.

If machine zero is the operand value, 32 zeros are returned as the result.

Data flag branch conditions:

Data flag bit 46: Result is indefinite.

52

Ceiling (32 Bits)

Half Word, Format #A
Subfunction: None

(R) to (T)

The #52 instruction transmits to 32-bit register T the nearest integer greater than or equal to the 32-bit floating-point operand in 32-bit register R. The integer is an unnormalized, 32-bit floating-point number with a positive exponent.

If R's exponent is positive, the operand is moved directly to T. If R's exponent is negative, the operation shifts the two's complement of the coefficient right, end-off, and increases the exponent by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, sign bits are extended on the left. The two's complement of the shifted coefficient with zero exponent is moved into 32-bit register T.

If machine zero is the operand value, 32 zeros are returned as the result.

Data flag branch conditions:

Data flag bit 46: Result is indefinite.

53

Significant Square Root (32 Bits)

Half Word, Format #A
Subfunction: None

(R) to (T)

The #53 instruction loads the square root of the 32-bit floating-point number in 32–bit register R into 32-bit register T.

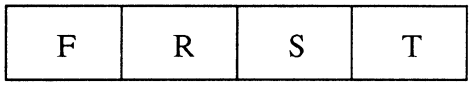
Data flag branch conditions:

Data flag bit 43:	Result is machine zero.
Data flag bit 45:	Square root result is imaginary.
Data flag bit 46:	Indefinite result.

54

Adjust Significance (32 Bits)

Half Word, Format #4
Subfunction: None



(R) per (S) to (T)

The #54 instruction adjusts the significance of the floating-point operand in 32-bit register R and transmits the adjusted result to 32-bit register T.

The right-most 24 bits of 32-bit register S contain a signed, two's complement integer. The absolute value of this integer is a shift count. If the shift count is positive, the operand's coefficient is shifted left the number of places specified by the shift count, or by the number of shifts needed to normalize the coefficient, whichever is smaller. The exponent of the operand is reduced by one for each place actually shifted. If the shift count is negative, the operation shifts the operand's coefficient to the right the number of specified places and increases the operand's exponent by one for each place shifted.

If R is indefinite, the result is indefinite, and data flag 46 is set. If R is machine zero, the result is machine zero, and data flag 43 is set. The instruction is undefined if the absolute value of the shift count is greater than 23, decimal.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Result is indefinite.

55

Adjust Exponent (32 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) per (S) to (T)

The #55 instruction moves the adjusted operand from 32-bit register R to 32-bit register T. The result's exponent is set equal to the exponent of the operand in 32-bit register S. The result's coefficient is formed by shifting the coefficient of the operand in R. If the R coefficient is zero, the exponent from S is copied to T with an all-zero coefficient.

The shift count used is the difference between the exponents in 32-bit registers R and S. If the R exponent is greater than the S exponent, a left shift is performed. A right shift occurs if the R exponent is less than S's exponent.

If the left shift count 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 of the operands are machine zero or indefinite, the result is set to indefinite, data flag bit 46 is set, and data flag bit 42 is clear.

Data flag branch conditions:

Data flag bit 42:	Excessive shift count; result is set to indefinite.
Data flag bit 46:	One or both operands are indefinite or machine zero; result is set to indefinite.

56

Select Link

Half Word, Format #7
Subfunction: 000ii000
Qualifiers: i=[ra,rb]

The #56 instruction combines the two vector operations that follow it into one single operation, by chaining output from the first vector instruction (*instr1*) to one of the inputs for the second vector instruction (*instr2*). Except when the R designator is zero, the #56 instruction must be immediately followed by the two vector instructions to be linked, otherwise the instruction is undefined. Table 1-1 lists the vector instructions that can be used in a link operation. The #56 instruction is undefined if *instr1* and *instr2* belong to the same unit.

Table 1-1. Vector instructions that can be used in a Link operation.

Unit	Instr1 Opcode	Instr2 Opcode
1	8A	8A
2	9D	9D
3	88,89,8B	88,89,8B
4	80,81,82,83,84,85, 86,87,90,91,92	80,81,82,83,84,85 86,87,90,91,92 C4,C5,C6,C7

Qualifiers *ra* and *rb* (Bits 3 and 4 of the R field) define the input to the second vector instruction, instr2. The *ra* qualifier specifies that *instr1*'s result will be the A input vector to instr2, and *rb* specifies that it will be the B input vector to *instr2*.

The linked instructions must observe certain conventions for their G-bit settings. The *h* qualifier must be the same in both instructions. However, each instruction can specify its own sign control qualifiers. For *instr1*, the *z* and *o* qualifiers, and Z and C operands are ignored, but for *instr2*, they specify the output vector. Data Flag bit results are the same as if both instructions ran as separate instructions.

Between the two linked instructions there can be two input vectors (A and B) and at least one broadcast value, or one input vector and two broadcast values. The *ra* and *rb* qualifiers for the #56 instruction, and the *a* and *b* qualifiers on the linked vector instructions, determine the input vectors and broadcast values that can be selected. Valid combinations are listed in table 1-2.

R-bits 0-2 and 5-7 are undefined and must be cleared to zero.

Table 1-2: Valid Combinations for Linked Vector Instructions.

Qual-ifiers	Instr1 qual.	Instr2 qual.	Instr1		Instr2	
			Input A	Input B	Input A	Input B
ra	none	*b*	vector A1	vector B1	instr1 output	broadcast B2
rb	none	*a*	vector A1	vector B1	broadcast A2	instr1 output
ra	*b*	none	vector A1	broadcast B1	instr1 output	vector B2
rb	*a*	none	broadcast A1	vector B1	vector A2	instr1 output
ra	*a*	*b*	broadcast A1	vector B1	instr1 output	broadcast B2
rb	*a*	*a*	broadcast A1	vector B1	broadcast A2	instr1 output
ra	*b*	*b*	vector A1	broadcast B1	instr1 output	broadcast B2
rb	*b*	*a*	vector A1	broadcast B1	broadcast A2	instr1 output
ra&rb	none	none	vector A1	vector B1	instr1 output	instr1 output
ra&rb	*a*	none	broadcast A1	vector B1	instr1 output	instr1 output
ra&rb	*b*	none	vector A1	broadcast B1	instr1 output	instr1 output
none			No linking takes place. Instr1 and Instr2 are separate operations.			

57

Read Domain Registers

Half Word, Format #7
Subfunction: None

Special Register per R to 64-Bit (T)

The #57 instruction reads the domain register specified by the R designator, and transmits its value to an area in register T. The specified register quantity definitions correspond to the R designator value:

R Designator Value	Source	Register T Bits
00	Stack Index	48–60
02	Previous Domain Package Number	52–58
03	Current Domain Package Number	52–58

The unspecified bits of register T are zeros for the defined values of R.

In Job mode, an undefined value in the R designator produces undefined results in register T. In Monitor mode, the instruction will always produce undefined results in register T.

58

Transmit Operand (32 Bits)

Half Word, Format #A
Subfunction: None

(R) to (T)

The #58 instruction transmits the 32-bit operand in 32-bit register R to 32-bit register T.

59

Transmit Absolute (32 Bits)

Half Word, Format #A
Subfunction: None

Absolute (R) to (T)

The #59 instruction transmits the absolute value of the 32-bit floating-point number in 32-bit register R to 32-bit register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

5A

Transmit Exponent (32 Bits)

Half Word, Format #A
Subfunction: None

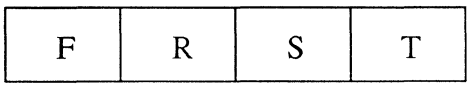
Exponent (R) to (T)

The #5A instruction transmits the exponent from the left-most 8 bits of 32-bit register R to the right-most 8 bits of 32-bit register T. The exponent's sign is extended through bit 8 of register T. The left-most 8 bits of register T are cleared to zero.

5B

Pack (32 Bits)

Half Word, Format #4
Subfunction: None



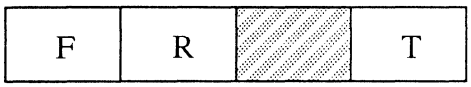
(R), (S) to (T)

The #5B instruction transmits a 32-bit floating-point number to 32-bit register T. The number's exponent is obtained from the right-most 8 bits of 32-bit register R, and its coefficient from the right-most 24 bits of 32-bit register S.

5C

Extend

Half Word, Format #A
Subfunction: None



32-Bit (R) to 64-Bit (T)

The #5C instruction extends a 32-bit floating-point number in 32-bit register R into a 64-bit floating-point number, and stores it in 64-bit register T.

The value of the resulting 16-bit exponent is 24 less than that of the source's exponent. The coefficient is obtained by transmitting the right-most 24 bits of register R into bits 16–39 of register T. The right-most bits of register T are cleared to zero.

If register R is indefinite, register T is indefinite, and data flag 46 is set. If register R is machine zero, register T is machine zero, and data flag 43 is set.

Data flag branch conditions:

Data flag bit 43:	Result machine zero.
Data flag bit 46:	Indefinite result.

5D

Index Extend

Half Word, Format #A
Subfunction: None

32-Bit (R) to 64-Bit (T)

The #5D instruction extends a 32-bit floating-point number in 32–bit register R into a 64-bit floating-point number, and stores it in 64–bit register T.

The resulting 16-bit exponent is the same value as the source's exponent. The coefficient is obtained by moving the right-most 24 bits of register R to bits 40–63 of register T. Bits 16–39 of register T are set to the sign of the source coefficient.

If register R is indefinite, register T is indefinite, and data flag 46 is set. If register R is machine zero, register T is machine zero, and data flag 43 is set.

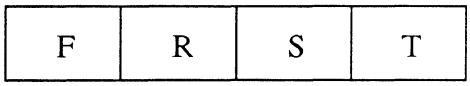
Data flag branch conditions:

Data flag bit 43:	Result machine zero.
Data flag bit 46:	Indefinite result.

5E

Load; Halfword

Half Word, Format #7
Subfunction: None



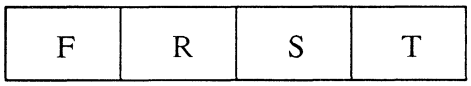
(T) per (S), (R)

The #5E instruction loads the contents of the 32-bit register T from the CP memory address formed by adding the contents of the 64-bit registers R and S. Register R contains the absolute base address, and register S contains an item count in half words that is left-shifted 5 bits before the addition. Any overflow from this addition is ignored.

5F

Store; Halfword

Half Word, Format #7
Subfunction: None



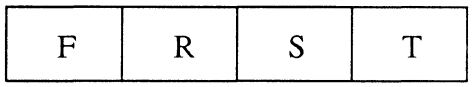
(T) per (S), (R)

The #5F instruction stores the contents of the 32-bit register T into the CP at the memory address formed by adding the contents of the 64-bit registers R and S. Register R contains the absolute base address, and register S contains an item count in half words that is left-shifted 5 bits before the addition. Any overflow from this addition is ignored.

60

Add; Upper Result (64 Bits)

Half Word, Format #4
Subfunction: None



(R) + (S) to (T)

The #60 instruction performs floating-point addition on the contents of the 64-bit registers R and S, returning the upper result in register T.

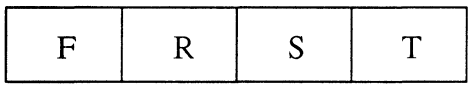
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

61

Add; Lower Result (64 Bits)

Half Word, Format #4
Subfunction: None



(R) + (S) to (T)

The #61 instruction performs floating-point addition on the contents of the 64-bit registers R and S, returning the lower result in register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

62

Add; Normalized Result (64 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) + (S) to (T)

The #62 instruction performs floating-point addition on the contents of the 64-bit registers R and S, returning the normalized upper result in register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

63

Add Address

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) + (S) to (T)

The #63 instruction adds bits 16–63 of register R to bits 16–63 of register S, storing the result in bits 16–63 of register T. Bits 16–63 are treated as unsigned, positive integers. Arithmetic overflow is ignored. Bits 0–15 of R are transferred without modification to bits 0–15 of register T.

64

Subtract; Upper Result (64 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) – (S) to (T)

The #64 instruction performs floating-point subtraction on the contents of the 64-bit registers R and S, returning the upper result in register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

65

Subtract; Lower Result (64 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) – (S) to (T)

The #65 instruction performs floating-point subtraction on the contents of the 64-bit registers R and S, returning the lower result in register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

66

Subtract; Normalized Result (64 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) - (S) to (T)

The #66 instruction performs floating-point subtraction on the contents of the 64-bit registers R and S, returning the normalized upper result in register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

67

Subtract Address

Half Word, Format #4
Subfunction: None

F	R	S	T

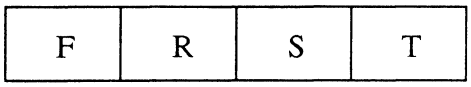
(R) - (S) to (T)

The #67 instruction subtracts bits 16-63 of register S from bits 16-63 of register R, storing the result in bits 16-63 of register T. Bits 16-63 are treated as 48-bit unsigned, positive integers. Arithmetic overflow is ignored. Bits 0-15 of R are transferred without modification to bits 0-15 of register T.

68

Multiply; Upper Result (64 Bits)

Half Word, Format #4
Subfunction: None



(R) * (S) to (T)

The #68 instruction performs floating-point multiplication on the contents of the 64-bit registers R and S, returning the upper result in register T.

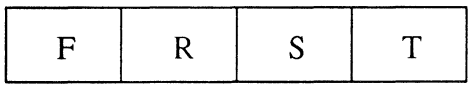
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

69

Multiply; Lower Result (64 Bits)

Half Word, Format #4
Subfunction: None



(R) * (S) to (T)

The #69 instruction performs floating-point multiplication on the contents of the 64-bit registers R and S, returning the lower result in register T.

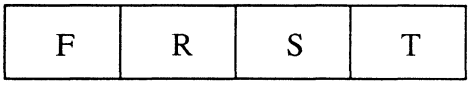
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

6B

Multiply; Significant Result (64 Bits)

Half Word, Format #4
Subfunction: None



(R) * (S) to (T)

The #6B instruction performs floating-point multiplication on the contents of the 64-bit registers R and S, returning the significant result in register T.

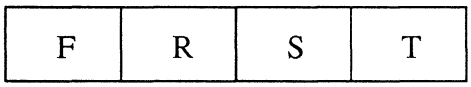
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

6C

Divide; Upper Result (64 Bits)

Half Word, Format #4
Subfunction: None



(R) / (S) to (T)

The #6C instruction performs floating-point division on the contents of the 64-bit registers R and S, returning the upper result in register T.

Data flag branch conditions:

Data flag bit 41:	Floating-point divide fault
Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

6D

Insert Bits (64 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) to (T) per (S)

The #6D instruction inserts the right-most bits of register R into register T. Bits 10–15 of register S specify the number of right-most bits to insert. The right-most 6 bits of S specify the beginning bit position in T of the inserted bits. Bits 0–9 and 16–57 of S are undefined, and must be zero. If the R designator is zero, register #00 provides machine zero.

The result is undefined if the number of inserted bits is zero, or if the number of inserted bits plus the beginning bit position in T exceeds 64.

6E

Extract Bits (64 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) to (T) per (S)

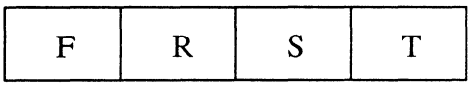
The #6E instruction extracts a specified number of bits from register R into the right-most portion of register T. Register T is cleared before receiving the bits. Bits 10–15 of register S specify the number of bits to extract from register R. The right-most 6 bits of S specify the left-most bit position in R of the extracted bits. Bits 0–9 and 16–57 of S are undefined, and must be zero. If the R designator is zero, register #00 provides machine zero.

The result of this instruction is undefined if the number of extracted bits is zero, or if the number of extracted bits plus the beginning bit position in R exceeds 64.

6F

Divide; Significant Result (64 Bits)

Half Word, Format #4
Subfunction: None



(R) / (S) to (T)

The #6F instruction performs a floating-point divide significant operation on the contents of the 64-bit registers R and S, returning the significant result in register T.

Data flag branch conditions:

Data flag bit 41:	Floating-point divide fault
Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

70

Truncate (64 Bits)

Half Word, Format #A
Subfunction: None

(R) to (T)

The #70 instruction transmits to register T the nearest integer whose magnitude is less than or equal to magnitude of the 64-bit floating-point operand in register R. The integer is an unnormalized, 64-bit floating-point number with a positive exponent.

If R's exponent is positive, the operand is moved directly to T. If R's exponent is negative, the magnitude of the coefficient is shifted right, end-off, and the exponent increased by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, zeros are extended on the left. If R's coefficient is positive, the shifted coefficient with zero exponent is moved into register T. If the coefficient is negative, the two's complement of the shifted coefficient, with zero exponent, is moved.

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

Data flag branch conditions:

Data flag bit 46: Indefinite result.

71

Floor (64 Bits)

Half Word, Format #A
Subfunction: None

(R) to (T)

The #71 instruction transmits to register T the nearest integer less than or equal to the 64-bit floating-point operand in register R. The integer is an unnormalized, 64-bit floating-point number with a positive exponent.

If R's exponent is positive, the operand is moved directly to T. If R's exponent is negative, the coefficient is shifted right, end-off, and the exponent increased by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, sign bits are extended on the left. The shifted coefficient with zero exponent is moved into register T.

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

Data flag branch conditions:

Data flag bit 46: Indefinite result.

72

Ceiling (64 Bits)

Half Word, Format #A
Subfunction: None

(R) to (T)

The #72 instruction transmits to register T the nearest integer greater than or equal to the 64-bit floating-point operand in register R. The integer is an unnormalized, 64-bit floating-point number with a positive exponent.

If R's exponent is positive, the operand is moved directly to T. If R's exponent is negative, the two's complement of the coefficient is shifted right, end-off, and the exponent increased by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, sign bits are extended on the left. The two's complement of the shifted coefficient with zero exponent is moved into register T.

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

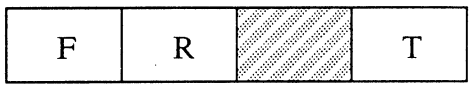
Data flag branch conditions:

Data flag bit 46: Indefinite result.

73

Significant Square Root (64 Bits)

Half Word, Format #A
Subfunction: None



(R) to (T)

The #73 instruction loads the square root of the 64-bit floating-point number in register R into 64-bit register T.

Data flag branch conditions:

Data flag bit 43:	Result is machine zero.
Data flag bit 45:	Square root result is imaginary.
Data flag bit 46:	Indefinite result.

74

Adjust Significance (64 Bits)

Half Word, Format #4
Subfunction: None

F	R	S	T

(R) per (S) to (T)

The #74 instruction adjusts the significance of the floating-point operand in register R and transmits the result to register T.

The right-most 48 bits of register S contain a signed, two's complement integer. The absolute value of this integer is a shift count. If the shift count is positive, the operand's coefficient is shifted 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 operand's exponent is reduced by one for each place shifted. An all-zero coefficient is shifted left the number of specified places.

If the shift count is negative, the operand's coefficient is shifted right the number of specified places. The operand's exponent is increased by one for each place shifted. The instruction is undefined if the absolute value of the shift count is greater than 47 decimal.

If R is indefinite, the result is indefinite, and data flag 46 is set. If R is machine zero, the result is machine zero, and data flag 43 is set.

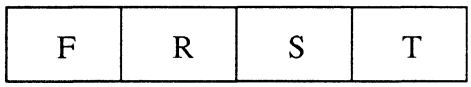
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Result is indefinite.

75

Adjust Exponent (64 Bits)

Half Word, Format #4
Subfunction: None



(R) per (S) to (T)

The #75 instruction moves the adjusted operand from register R into register T. The result's exponent is set equal to the exponent of the operand in register S. The result is formed by shifting the coefficient of the operand in R. If the R coefficient is zero, the exponent from S is copied to T with an all-zero coefficient.

The shift count used is the difference between the exponents in registers R and S. If the R exponent is greater than the S exponent, a left shift is performed. A right shift occurs if the R exponent is less than the S exponent.

If the left shift count exceeds that required to normalize the coefficient in register R, the result is set to indefinite, and data flag bit 42 is set. If either or both of the operands are machine zero or indefinite, the result is set to indefinite, data flag bit 46 is set, and data flag bit 42 is clear.

Data flag branch conditions:

Data flag bit 42:	Excessive shift count; result is set to indefinite.
Data flag bit 46:	One or both operands are indefinite or machine zero; result is set to indefinite.

76

Contract

Half Word, Format #A
Subfunction: None

64-Bit (R) to 32-Bit (T)

The #76 instruction contracts the 64-bit floating-point number in register R into a 32-bit floating-point number. The 32-bit result is transmitted to register T. The 24-bit result coefficient is copied from left-most 24 bits (bits 16–39) of the source coefficient in R. This has the effect of contracting to minus one all negative source coefficients whose absolute values (neglecting the exponent) were less than or equal to 2^{24}.

The exponent of the operand from register R is increased by 24 as it is moved to register T. The resultant exponent generated from different values of input exponents is as follows:

Input Exponent	Result Exponent
7FFF...7000	Result indefinite - Data Flag bit 46.
6FFF...0058	Result indefinite - Data Flag bits 42 and 46.
0057...FF78	Result exponent 24 larger than the input exponent.
FF77...8000	Result machine zero - Data Flag bit 43.

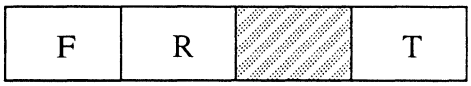
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Result is indefinite.

77

Rounded Contract

Half Word, Format #A
Subfunction: None



64-Bit (R) to 32-Bit (T)

The #77 instruction performs a rounded contract operation on the 64-bit floating-point number in register R, and transmits the 32-bit 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 shifted right one place. The left-most 24 bits of the 48-bit sum are transmitted to the 24-bit coefficient part of register T. Each non-endcase result element's 8-bit exponent is 24 (25 if overflow occurred) greater than the corresponding source element's exponent.

If the input operand is between #FF77 and #8000, the result is machine zero, even if the rounding operation would take it out of machine zero.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Result is indefinite.

78

Transmit Operand (64 Bits)

Half Word, Format #A
Subfunction: None

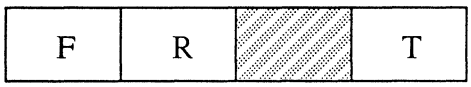
(R) to (T)

The #78 instruction transmits the 64-bit operand in register R to register T.

79

Transmit Absolute (64 Bits)

Half Word, Format #A
Subfunction: None



(R) to (T)

The #79 instruction transmits the absolute value of the 64-bit floating-point number in register R to register T.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Result is indefinite.

7A

Transmit Exponent (64 Bits)

Half Word, Format #A
Subfunction: None

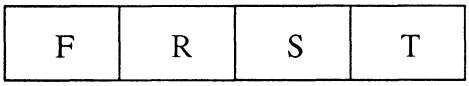
(R) to (T)

The #7A instruction transmits the exponent from the left-most 16 bits of register R to the right-most 16 bits of register T. The exponent's sign is extended through bit 16 of register T. The left-most 16 bits of register T are cleared to zero.

7B

Pack (64 Bits)

Half Word, Format #4
Subfunction: None



(R), (S) to (T)

The #7B instruction transmits a 64-bit floating-point number to register T. The number's exponent is obtained from the right-most 16 bits of register R, and its coefficient from the right-most 48 bits of register S.

7C

Transmit Length (64 Bits)

Half Word, Format #A
Subfunction: None

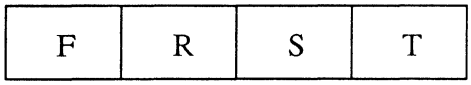
(R) to (T)

The #7C instruction transmits the left-most 16 bits of register R to the right-most 16 bits of register T. The left-most 48 bits of register T are cleared to zero.

7D

Swap

Half Word, Format #7
Subfunction: None



S ---> T and R ---> S

The #7D instruction moves part of the register file to CP memory at the destination addressed by register T. The move begins with the 64-bit register specified by the right-most 8 bits of register S. The operation then transmits the source field R from CP memory to the register file, beginning at the 64-bit register specified by the right-most 8 bits of register S. Register S must specify an even numbered register.

The left-most 16 bits of registers R and T specify the field length in words for the source and destination fields respectively. Although the source and destination field lengths may be different, each must be an even number. A zero field length means that no transfer is to occur for that field. Any transfer in or out of the register file that exceeds the register file's limits causes the instruction to be undefined.

The right-most 48 bits of registers R and T specify the source and destination base address respectively. The address must be a 64-bit word in CP memory on an even word boundary.

Bits 57–63 in registers R and T are undefined, and must be zero. Overlapping source and destination fields are allowed only if the base addresses for both fields are equal. The operand registers R, S, and T can be in the range of the registers being swapped.

This instruction is illegal if the attempt is to transfer a vector with an odd starting address, or of odd length.

7E

Load Word

Half Word, Format #7
Subfunction: None

F	R	S	T

(T) per (S), (R)

The #7E instruction loads the contents of register T from the CP memory address specified by R + S. Register S contains an item count in words that is shifted left 6 places, then added to the base address in R. Overflow is ignored.

7F

Store Word

Half Word, Format #7
Subfunction: None

F	R	S	T

(T) per (S), (R)

The #7F instruction stores the contents of register T into the CP memory address specified by registers R + S. Register S contains an item count in words that is shifted left 6 places, then added to the base address in R. Overflow is ignored.

80

Add; Upper Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers: h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]

F	G	X	A	Y	B	Z	C

A + B ---> C

The #80 instruction performs floating-point addition on the elements of vectors A and B, storing the upper result in the corresponding elements of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

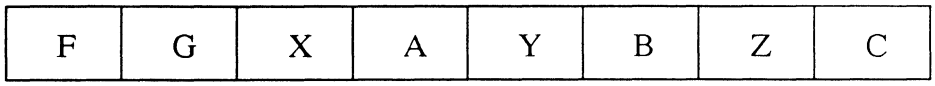
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

81

Add; Lower Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers: h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A + B ---> C

The #81 instruction performs floating-point addition on the elements of vectors A and B, storing the lower result in the corresponding elements of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

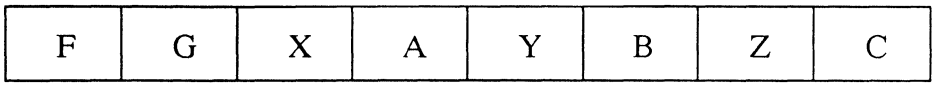
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

82

Add; Normalized Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers: h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A + B ---> C

The #82 instruction performs floating-point addition on the elements of vectors A and B, storing the normalized upper result in the corresponding elements of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

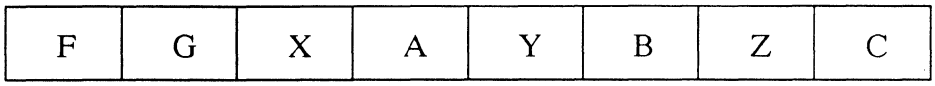
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

83

Add Address

Full Word, Format #1
Subfunction: 0zoab000
Qualifiers: z,o,a,b



A + B ---> C

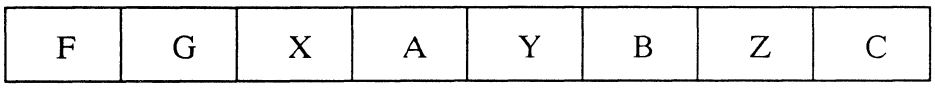
The #83 instruction adds bits 16–63 of elements of vector B to bits 16–63 of elements of vector A. The results are stored in bits 16–63 of each vector C element. Results are treated as 48-bit, positive, unsigned integers. Arithmetic overflow is ignored. The left-most 16 bits of each element of vector A are transferred without modification to the left-most 16 bits of the corresponding element of vector C.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B.

84

Subtract; Upper Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers: h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A - B ---> C

The #84 instruction performs floating-point subtraction of the elements of vectors A and B. To subtract, the coefficient part of vector B is complemented as in two's complement arithmetic, and the result added to vector A. The upper result is stored in the corresponding element of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

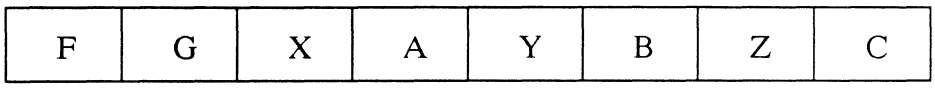
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

85

Subtract; Lower Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers: h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A - B ---> C

The #85 instruction performs floating-point subtraction of the elements of vectors A and B. To subtract, the coefficient part of vector B is complemented as in two's complement arithmetic, and the result added to vector A. The lower result is stored in the corresponding element of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

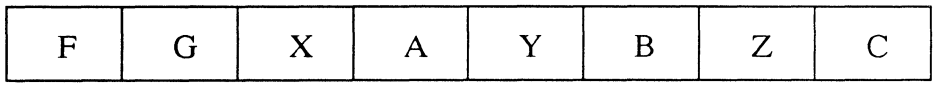
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

86

Subtract; Normalized Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers : h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A – B ---> C

The #86 instruction performs floating-point subtraction of the elements of vectors A and B. To subtract, the coefficient part of vector B is complemented as in two's complement arithmetic, and the result added to vector A. The normalized upper result is stored in the corresponding element of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

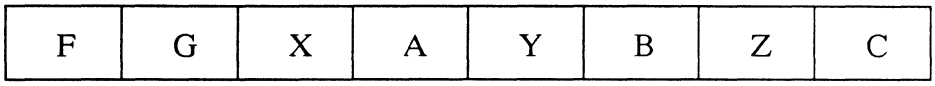
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

87

Subtract Address

Full Word, Format #1
Subfunction: 0zoab000
Qualifiers: z,o,a,b



A - B ---> C

The #87 instruction subtracts bits 16–63 of elements of vector B from bits 16–63 of elements of vector A. Vector B is complemented as in two's complement arithmetic, and the result added to vector A. The results are stored in bits 16–63 of each vector C element. Bits 16–63 are treated as positive, unsigned integers. Arithmetic overflow is ignored. The left-most 16 bits of each element of vector A are transferred without modification to the left-most 16 bits of the corresponding vector C element.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B.

88

Multiply; Upper Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers : h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]

F	G	X	A	Y	B	Z	C

A * B ---> C

The #88 instruction performs floating-point multiplication of vector A's elements by those of vector B. The upper part of the result is stored in the corresponding element of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

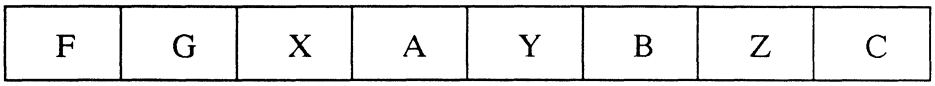
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

89

Multiply; Lower Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers : h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A * B ---> C

The #89 instruction performs floating-point multiplication of vector A's elements by those of vector B. The lower part of the result is stored in the corresponding element of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

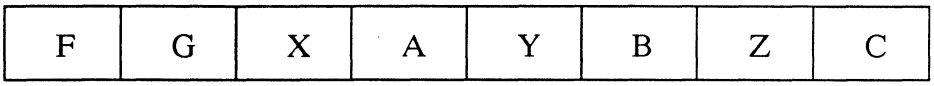
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

8A

Shift Element

Full Word, Format #1
Subfunction: 0zoab000
Qualifiers: z,o,a,b



A per B ---> C

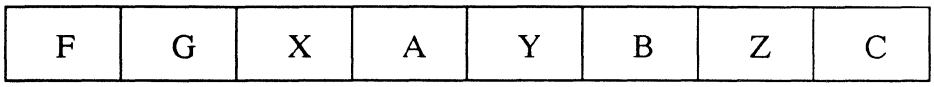
The #8A instruction shifts each 64-bit element of vector A left or right, as specified by the corresponding element of vector B. The result is stored in the corresponding element of vector C. The 8-bit signed integer in the right-most byte of the vector B element specifies the shift count. For positive integers between #00 to #3F, the vector A element is shifted left end-around for the specified number of places. For negative integers between #FF and #C1, the element is shifted right with sign bit extension. Bit 0 in each vector A operand is the sign bit for the extension. The number of right shifts performed is the two's complement of the right-most bytes of the operands in vector B. If the absolute value of the shift count is greater than #3F or less than #C1, the results are undefined. The left-most 7 bytes of vector B elements are ignored.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B.

8B

Multiply; Significant Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers: h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A * B ---> C

The #8B instruction performs floating-point multiplication of elements of vector A by those of vector B. The significant part of the floating-point result is stored in the corresponding element of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

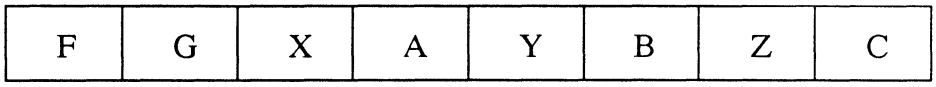
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

8C

Divide; Upper Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers : h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A / B ---> C

The #8C instruction performs floating-point division of vector A's elements by corresponding elements of vector B. The upper part of the result is stored in the corresponding element of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

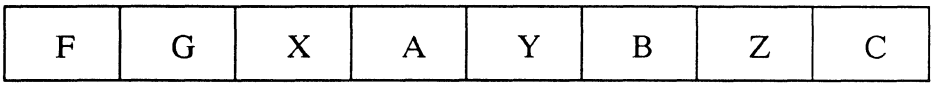
Data flag branch conditions:

Data flag bit 41:	Floating-point divide fault.
Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

8F

Divide; Significant Result

Full Word, Format #1
Subfunction: hzoabsss
Qualifiers : h,z,o,a,b,sss=[ma,c,(n=ma+c),mb]



A / B ---> C

The #8F instruction performs floating-point division of vector A's elements by corresponding elements of vector B. The significant part of the floating-point result is stored in the corresponding element of vector C. Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B. The sign control feature is valid for this instruction. The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2.

Data flag branch conditions:

Data flag bit 41:	Floating-point divide fault.
Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

90

Truncate

Full Word, Format #1
Subfunction: hzoa0000
Qualifiers: h,z,o,a

A ---> C

The #90 instruction transmits to vector C 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 an unnormalized floating-point number with a positive exponent. If machine zero is the operand value, the result element is all-zero. All elements are 32 or 64-bit floating-point operands, depending on the *h* qualifier.

If the vector A element's exponent is positive, the element is moved directly to vector C. If the exponent is negative, the magnitude of the coefficient is shifted right end-off, and the exponent increased by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, zeros are extended on the left. For positive coefficients, the shifted coefficient with zero exponent is moved into the vector C element. For negative coefficients, the two's complement of the shifted coefficient, with zero exponent, is moved.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bit 46). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A.

Data flag branch conditions:

Data flag bit 46: Indefinite result.

91

Floor

Full Word, Format #1
Subfunction: hzoa0000
Qualifiers: h,z,o,a

A ---> C

The #91 instruction transmits to vector C the nearest integer less than or equal to the corresponding floating-point element of source vector A. This integer is an unnormalized floating-point number with a positive exponent. If machine zero is the operand value, the resulting element is all-zero. All elements are 32 or 64-bit floating-point operands, depending on the *h* qualifier.

If the vector A element's exponent is positive, the element is moved directly to vector C. If the exponent is negative, the operation shifts the coefficient right end-off, and increases the exponent by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, sign bits are extended on the left. The shifted coefficient with zero exponent is moved into the vector C element.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bit 46). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A.

Data flag branch conditions:

Data flag bit 46: Indefinite result.

92

Ceiling

Full Word, Format #1
Subfunction: hzoa0000
Qualifiers: h,z,o,a

A ---> C

The #92 instruction transmits to vector C the nearest integer greater than or equal to the corresponding floating-point element of source vector A. The integer is an unnormalized floating-point number with a positive exponent. If machine zero is the operand value, the resulting element is all-zero. All elements are 32 or 64-bit floating-point operands, depending on the *h* qualifier.

If the vector A element's exponent is positive, the element is moved directly to vector C. If the exponent is negative, the two's complement of the coefficient is shifted right end-off, and the exponent increased by one for each bit position shifted, until the exponent is zero. As the coefficient is shifted, sign bits are extended on the left. The two's complement of the shifted coefficient, with zero exponent, is moved to the vector C element.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bit 46). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A.

Data flag branch conditions:

Data flag bit 46: Indefinite result.

93

Significant Square Root

Full Word, Format #1
Subfunction: hzoa0ss0
Qualifiers : h,z,o,a,ss=[ma,c]

A ---> C

The #93 instruction forms the square root of each element of vector A, and moves it into the corresponding element of result vector C. Elements of vectors A and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A. The sign control feature is valid for this instruction. The effect of the three qualifiers that control the state of subfunction bits 5 and 6, used for sign control, is discussed in chapter 2.

Data flag branch conditions:

Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 45:	Square root result is imaginary.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

94

Adjust Significance

Full Word, Format #1
Subfunction: hzoab000
Qualifiers: h,z,o,a,b

F	G	X	A	Y	B	Z	C

A per B ---> C

The #94 instruction adjusts the significance of the floating-point elements from vector A, and transmits the results to the corresponding elements of vector C. The instruction operates on 64-bit words, unless the *h* qualifier is specified. Elements of vector B contain signed two's complement integers in the right-most 48 (24 if the *h* qualifier is specified) bits. The absolute values of these integers are shift counts. The result is undefined if the absolute value of the shift count is greater than 47 (23 if the *h* qualifier is specified).

If the shift count is positive, the vector A element's coefficient is shifted 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 element's exponent is reduced by one for each place shifted. An all-zero coefficient is shifted left the number of specified positions. If the shift count is negative, the element's coefficient is shifted right the number of places specified by the shift count. The element's exponent is increased by one for each place shifted.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B.

If a vector A element is indefinite, the resulting vector C element is indefinite, and data flag 46 is set. If a vector A element is machine zero, the resulting vector C element is machine zero, and data flag 43 is set.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

95

Adjust Exponent

Full Word, Format #1
Subfunction: hzoab000
Qualifiers: h,z,o,a,b

F	G	X	A	Y	B	Z	C

A per B ---> C

The #95 instruction transmits adjusted elements from vector A to vector C. The exponent of a result element is set equal to the exponent of the associated vector B element. Result elements' coefficients are formed by shifting the coefficients of the vector A elements. The instruction operates on 64-bit words, unless the *h* qualifier is specified.

The shift count used is the difference between the exponents of associated elements from A and B. If a vector A element's exponent is greater than that of an element of vector B, the shift is to the left; a right shift is performed if the exponent is less. For vector A element coefficients that are zero, the vector B exponent 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 set. If either or both operands are indefinite or machine zero, the result is indefinite. Data flag 46 is set and 42 is not set in this case.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

96

Contract

Full Word, Format #1
Subfunction: 0zoa0000
Qualifiers: z,o,a

64-Bit A ---> 32-Bit C

The #96 instruction forms each 32-bit floating-point element of result vector C by contracting the corresponding 64-bit floating-point vector A element. Each non-endcase 8-bit result element's exponent is 24 greater than its source element's exponent. Each 24-bit result's coefficient is copied from the source coefficient's left-most 24 bits (bits 16–39). This has the effect of contracting to minus one all negative source coefficients whose absolute values (neglecting the exponent) were less than or equal to 2^{24}.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

97

Rounded Contract

Full Word, Format #1
Subfunction: 0zoa0000
Qualifiers: z,o,a

64-Bit A ---> 32-Bit C

The #97 instruction forms each 32-bit floating-point element of result vector C by performing a rounded contract operation on the corresponding 64-bit floating-point vector A element. A positive one is added to bit 40 of the origin operand. If overflow occurs, the exponent is increased by one, and the coefficient shifted right one place. This sum's left-most 24 bits are transmitted to the 24-bit coefficient part of result element C. Each non-endcase result element's 8-bit exponent is 24 (25 if overflow occurred) greater than the corresponding source element's exponent.

If the input operand is between #FF77 and #8000, the result is machine zero, even if the rounding operation would take it out of machine zero.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

98

Transmit Element

Full Word, Format #1
Subfunction: hzoa0000
Qualifiers: h, z,o,a

A ---> C

The #98 instruction transmits the source vector A to result vector C. All elements are 32 or 64-bit floating-point operands, depending on the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A.

99

Move Absolute

Full Word, Format #1
Subfunction: hzoa0000
Qualifiers: h,z,o,a

A ---> C

The #99 instruction moves the absolute value of each vector A floating-point element to the corresponding floating-point element in vector C. All elements are 32 or 64-bit floating-point operands, depending on the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

9A

Move Exponent

Full Word, Format #1
Subfunction: hzoa0000
Qualifiers: h,z,o,a

A ---> C

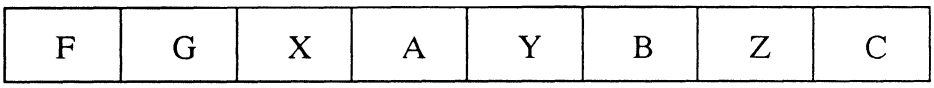
The #9A instruction forms vector C elements by storing exponents from the input vector A into the right-most portion of the coefficients of vector C elements. The exponent's sign is extended left to the coefficient sign bit position. Each vector C element's exponent portion is cleared to zero. All elements are 32 or 64-bit operands, depending on the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that register A contains a constant that is broadcast as the common value for elements of vector A.

9B

Pack

Full Word, Format #1
Subfunction:hzoab000
Qualifiers: h,z,o,a,b



A, B ---> C

The #9B instruction transmits to each result vector C element a 64 or 32-bit floating-point number produced as follows. The right-most 16 or 8 bit positions of each vector A element (as an exponent) are moved to the left-most 16 or 8 bit positions of result vector C, and the right-most 48 or 24 bits of each vector B element (the coefficient) are moved to the right-most 48 or 24 bits of result vector C. Elements of vectors A, B and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B.

9C

Extend

Full Word, Format #1
Subfunction:0zoa0000
Qualifiers: z,o,a

32-Bit A ---> 64-Bit C

The #9C instruction forms result vector C by extending 32-bit floating-point operands of vector A into 64-bit floating-point operands. The value of each resulting 16-bit exponent is 24 less than that of the corresponding source element's exponent. Each result coefficient is obtained by transmitting the right-most 24 bits of the corresponding source element into bits 16–39 of each result element. The right-most 24 bits of each result are cleared to zero.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifier *a* indicates that 32–bit register A contains a constant that is broadcast as the common value for elements of vector A. If an element of vector A is indefinite, the corresponding vector C element is set to indefinite, and data flag 46 set. If a vector A element is machine zero, machine zero is stored in the corresponding vector C element, and data flag 43 set.

Data flag branch conditions:

Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

9D

Logical Operation

Full Word, Format #1
Subfunction: hzoabnnn
Qualifiers: h,z,o,a,b, nnn=[000,001,010,011,100,101,110,111]

F	G	X	A	Y	B	Z	C

A, B ---> C

The #9D instruction performs a bit-by-bit logical operation between elements of vectors A and B. The result is transmitted to vector C. Elements of vectors A, B and C are 64 bits by default, or 32 bits by declaring the *h* qualifier. The logical operation performed depends on bits 5, 6, and 7 in the G field. The valid bit settings for each operation are:

000: Exclusive OR
001: AND
010: OR
011: NOT AND (stroke)
100: NOT OR (pierce)
101: OR NOT (implication)
110: AND NOT (inhibit)
111: Exclusive OR NOT (equivalence)

Table 1-1 describes the effect of each logical operation, depending on the bit settings in elements of A and B.

Table 1-1. Logical Operations on vector A and B elements.

Source A B	Excl. OR	AND	OR	NOT AND	NOT OR	OR NOT	AND NOT	Excl.OR NOT
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

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for elements of vectors A and B.

A0

Add; Upper Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers: h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]

F	G	X	A	Y	B	Z	C

A + B ---> C

The #A0 instruction performs floating-point addition on elements of sparse vectors A and B. The upper result is stored in the corresponding element of sparse vector C. Elements may be 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, machine zero is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg, ivg,* or *xvg* qualifier. Table 1-2 shows the logical function for each qualifier.

Table 1-2. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1	G-Bits 2	Qualifier	Logical Function Performed
0	0	None	Logical OR of X,Y
0	1	rvg	Logical AND of X,Y
1	0	xvg	Logical Exclusive OR of X,Y
1	1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-3.

Table 1-3. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

A1

Add; Lower Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]

F	G	X	A	Y	B	Z	C

A + B ---> C

The #A1 instruction performs floating-point addition on elements of sparse vectors A and B. The lower result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, machine zero is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg*, *ivg*, or *xvg* qualifier. Table 1-4 shows the logical function for each qualifier.

Table 1-4. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1	G-Bits 2	Qualifier	Logical Function Performed
0	0	None	Logical OR of X,Y
0	1	rvg	Logical AND of X,Y
1	0	xvg	Logical Exclusive OR of X,Y
1	1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-5.

Table 1-5. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0–15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

A2

Add; Normalized Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]

F	G	X	A	Y	B	Z	C

A + B ---> C

The #A2 instruction performs floating-point addition on elements of sparse vectors A and B. The normalized result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, machine zero is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg, ivg,* or *xvg* qualifier. Table 1-6 shows the logical function for each qualifier.

Table 1-6. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1 2	Qualifier	Logical Function Performed
0 0	None	Logical OR of X,Y
0 1	rvg	Logical AND of X,Y
1 0	xvg	Logical Exclusive OR of X,Y
1 1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-7.

Table 1-7. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

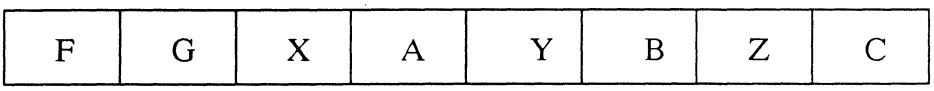
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

A4

Subtract; Upper Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]



A - B ---> C

The #A4 instruction performs floating-point subtraction on elements of sparse vectors A and B. The upper result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, machine zero is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg*, *ivg*, or *xvg* qualifier. Table 1-8 shows the logical function for each qualifier.

Table 1-8. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1 2	Qualifier	Logical Function Performed
0 0	None	Logical OR of X,Y
0 1	rvg	Logical AND of X,Y
1 0	xvg	Logical Exclusive OR of X,Y
1 1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-9.

Table 1-9. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

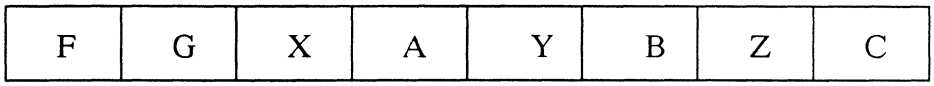
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

A5

Subtract; Lower Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]



A – B ---> C

The #A5 instruction performs floating-point subtraction on elements of sparse vectors A and B. The lower result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, machine zero is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg, ivg,* or *xvg* qualifier. Table 1–10 shows the logical function for each qualifier.

Table 1–10. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1 2	Qualifier	Logical Function Performed
0 0	None	Logical OR of X,Y
0 1	rvg	Logical AND of X,Y
1 0	xvg	Logical Exclusive OR of X,Y
1 1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-11.

Table 1-11. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

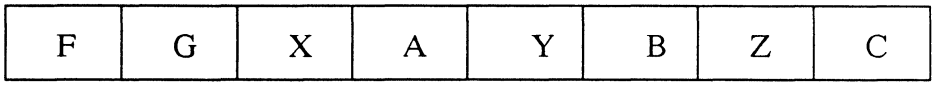
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

A6

Subtract; Normalized Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]



A - B ---> C

The #A6 instruction performs floating-point subtraction on elements of sparse vectors A and B. The normalized result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, machine zero is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg*, *ivg*, or *xvg* qualifier. Table 1-12 shows the logical function for each qualifier.

Table 1-12. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1 2	Qualifier	Logical Function Performed
0 0	None	Logical OR of X,Y
0 1	rvg	Logical AND of X,Y
1 0	xvg	Logical Exclusive OR of X,Y
1 1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-13.

Table 1-13. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

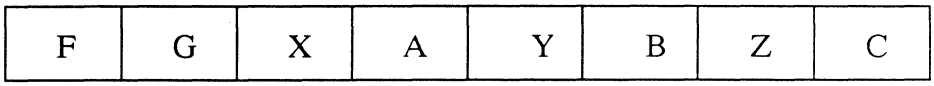
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

A8

Multiply; Upper Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]



A * B ---> C

The #A8 instruction performs floating-point multiplication on elements of sparse vectors A and B. The upper result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, normalized one is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg, ivg,* or *xvg* qualifier. Table 1-14 shows the logical function for each qualifier.

Table 1-14. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1 2	Qualifier	Logical Function Performed
0 0	None	Logical AND of X,Y
0 1	rvg	Logical OR of X,Y
1 0	xvg	Logical Exclusive OR of X,Y
1 1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-15.

Table 1-15. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

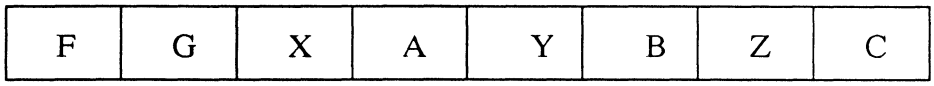
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

A9

Multiply; Lower Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]



A * B ---> C

The #A9 instruction performs floating-point multiplication on elements of sparse vectors A and B. The lower result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, normalized one is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg, ivg,* or *xvg* qualifier. Table 1-16 shows the logical function for each qualifier.

Table 1-16. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1 2	Qualifier	Logical Function Performed
0 0	None	Logical AND of X,Y
0 1	rvg	Logical OR of X,Y
1 0	xvg	Logical Exclusive OR of X,Y
1 1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-17.

Table 1-17. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

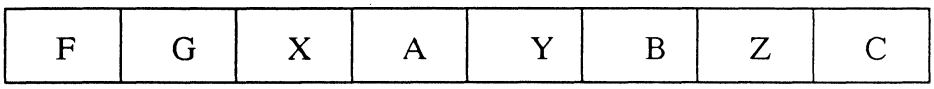
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

AB

Multiply; Significant Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]



A * B ---> C

The #AB instruction performs floating-point multiplication on elements of sparse vectors A and B. The significant result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, normalized one is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg, ivg,* or *xvg* qualifier. Table 1-18 shows the logical function for each qualifier.

Table 1-18. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1 2	Qualifier	Logical Function Performed
0 0	None	Logical AND of X,Y
0 1	rvg	Logical OR of X,Y
1 0	xvg	Logical Exclusive OR of X,Y
1 1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-19.

Table 1-19. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

AC

Divide; Upper Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers : h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]

F	G	X	A	Y	B	Z	C

A / B ---> C

The #AC instruction performs floating-point division on elements of sparse vectors A and B. The upper result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, normalized one is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg, ivg,* or *xvg* qualifier. Table 1-20 shows the logical function for each qualifier.

Table 1-20. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1	G-Bits 2	Qualifier	Logical Function Performed
0	0	None	Logical AND of X,Y
0	1	rvg	Logical OR of X,Y
1	0	xvg	Logical Exclusive OR of X,Y
1	1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-21.

Table 1-21. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

Data flag branch conditions:

Data flag bit 41:	Floating-point divide fault.
Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

AF

Divide; Significant Result

Full Word, Format #1
Subfunction: hllabsss
Qualifiers: h,ll=[rvg,xvg,ivg],a,b,sss=[ma,c,(n=ma+c),mb]

F	G	X	A	Y	B	Z	C

A / B ---> C

The #AF instruction performs floating-point division on elements of sparse vectors A and B. The significant result is stored in the corresponding element of sparse vector C. Elements are 64 bits by default, or 32 bits by declaring the *h* qualifier.

An element is read from sparse vector A whenever a one bit is encountered in the order vector X. When a one bit occurs in order vector Y, an element is read from vector B. If there is a zero bit in the order vector, normalized one is used as the associated A or B element.

Order vector Z is the result of a bit-by-bit logical function performed on order vectors X and Y, as specified by the selected *rvg, ivg,* or *xvg* qualifier. Table 1-22 shows the logical function for each qualifier.

Table 1-22. Logical Functions on X and Y to Produce Order Vector Z.

G-Bits 1 2	Qualifier	Logical Function Performed
0 0	None	Logical AND of X,Y
0 1	rvg	Logical OR of X,Y
1 0	xvg	Logical Exclusive OR of X,Y
1 1	ivg	Logical OR NOT of X,Y

The sparse vector C receives non-zero values corresponding to each one bit in the order vector Z, as defined in table 1-23.

Table 1-23. Results of the logical operations performed by the source vectors.

Source				Results							
Order Vector		Sparse Data Vector Element		G Bit 1 = 0 G Bit 2 = 0 OR		G Bit 1 = 0 G Bit 2 = 1 AND		G Bit 1 = 1 G Bit 2 = 0 Exclusive OR		G Bit 1 = 1 G Bit 2 = 1 Implication	
X	Y	A	B	Z	C	Z	C	Z	C	Z	C
0	0	MZ	MZ	0	N	0	N	0	N	1	MZ
0	1	MZ	B	1	+B	0	N	1	+B	0	N
1	0	A	MZ	1	A	0	N	1	A	1	A
1	1	A	B	1	A+B	1	A+B	0	N	1	A+B

Notes:

A	A stream operand
B	B stream operand
N	No result produced
MZ	Machine zero

For each one bit in order vector Z, an output element of vector C is generated. Vector C's length is moved to bits 0-15 of register C.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as the common value for an element of vector A and B. Either qualifier or both may be used. The sign control feature is valid for this instruction. The effect of the qualifiers which control the state of subfunction bits used for sign control are discussed in chapter 2.

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

Data flag branch conditions:

Data flag bit 41:	Floating-point divide fault.
Data flag bit 42:	Exponent overflow.
Data flag bit 43:	A result element in vector C is machine zero.
Data flag bit 46:	A result element of vector C is set to indefinite due to an input element being indefinite or exponent overflow.

B0

Compare Integers, Branch if Equal

Full Word, Format #C
Subfunction: h00fubb0
Qualifiers: h, f=[fwc],u=[usi],bb=[brf,brb,rel]

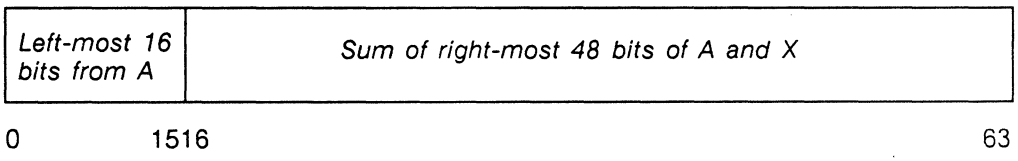
F	G	X	A	Y	B	Z	C

(A) + (X) EQ (Z)

The #B0 instruction executes as a Compare Integer and Branch operation when bits 1 and 2 in the G designator are zero. The two operands from register A and X are added, their sum compared to the integer in register Z, the sum of A and X are then transmitted to register C, and a branch taken according to the compare result.

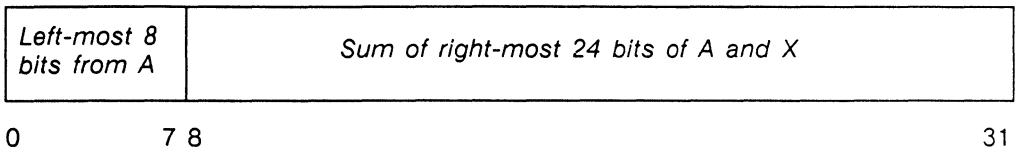
If the *h* qualifier is specified, the A, X, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to register Z's contents according to the *fwc* and *usi* qualifiers. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

If *fwc* is specified, 64 bits of the result stored to register C are compared to 64 bits previously read from register Z, otherwise the addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

If the comparison is not met, execution continues at the next sequential instruction.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if both *h* and *fwc* are specified, or if both qualifiers *brf* and *brb* are specified together.

B1

Compare Integers, Branch if Not Equal

Full Word, Format #C
Subfunction: h00fubb0
Qualifiers : h,f=[fwc],u=[usi],bb=[brf,brb,rel]

F	G	X	A	Y	B	Z	C

(A) + (X) NE (Z)

The #B1 instruction executes as a Compare Integer and Branch operation when bits 1 and 2 in the G designator are zero. The two operands from A and X are added, their sum compared to the integer in Z, the sum of A and X are then transmitted to register C, and a branch taken according to the result.

If the *h* qualifier is specified, the A, X, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

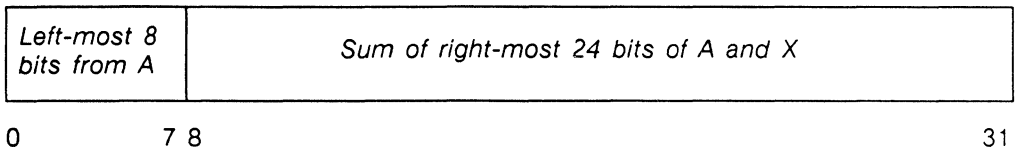
If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:

Left-most 16 bits from A	*Sum of right-most 48 bits of A and X*

0 1516 63

If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to register Z's contents according to the *fwc* and *usi* qualifiers. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

If *fwc* is specified, 64 bits of the result stored to register C are compared to 64 bits previously read from register Z, otherwise the addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

If the comparison is not met, execution continues at the next sequential instruction.

If the comparison is met, the instruction branches according to the specified qualifiers *rel, brf,* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

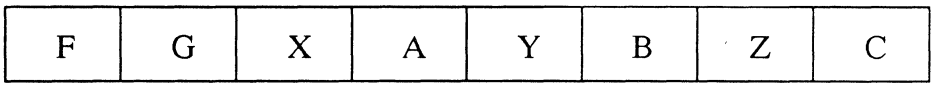
The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if both *h* and *fwc* are specified, if the qualifiers *brf* and *brb* are specified together.

B2

Compare Integers, Branch if Greater or Equal

Full Word, Format #C
Subfunction: h000ubb0
Qualifiers: h, u=[usi],bb=[brf,brb,rel]

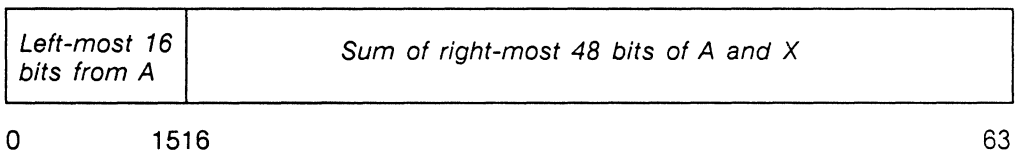


(A) + (X) GE (Z)

The #B2 instruction executes as a Compare Integer and Branch operation when bits 1 and 2 in the G designator are zero. The two operands from A and X are added, their sum compared to the integer in Z, the sum of A and X are then transmitted to register C, and a branch taken according to the result.

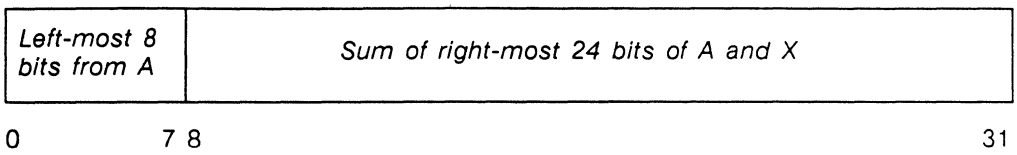
If the *h* qualifier is specified, the A, X, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to register Z's contents according to the *usi* qualifier. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

The addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

If the comparison is not met, execution continues at the next sequential instruction.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are specified together.

B3

Compare Integers, Branch if Less

Full Word, Format #C
Subfunction: h000ubb0
Qualifiers: h,u=[usi],bb=[brf,brb,rel]

F	G	X	A	Y	B	Z	C

(A) + (X) LT (Z)

The #B3 instruction executes as a Compare Integer and Branch operation when bits 1 and 2 in the G designator are zero. The two operands from A and X are added, their sum compared to the integer in Z, the sum of A and X are then transmitted to register C, and a branch taken according to the result.

If the *h* qualifier is specified, the A, X, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

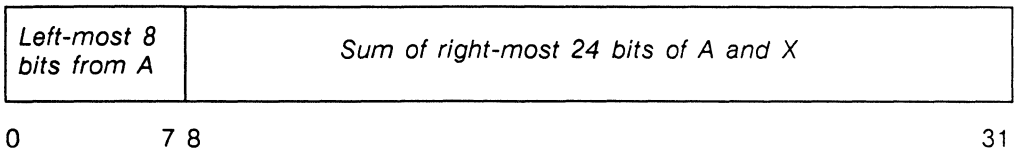
If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:

Left-most 16 bits from A	*Sum of right-most 48 bits of A and X*

0 1516 63

If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to register Z's contents according to the *usi* qualifier. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

The addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

If the comparison is not met, execution continues at the next sequential instruction.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are specified together.

B4

Compare Integers, Branch if Less or Equal

Full Word, Format #C
Subfunction: h000ubb0
Qualifiers: h,u=[usi],bb=[brf,brb,rel]

F	G	X	A	Y	B	Z	C

(A) + (X) LE (Z)

The #B4 instruction executes as a Compare Integer and Branch operation when bits 1 and 2 in the G designator are zero. The two operands from A and X are added, their sum compared to the integer in Z, the sum of A and X are then transmitted to register C, and a branch taken according to the result.

If the *h* qualifier is specified, the A, X, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

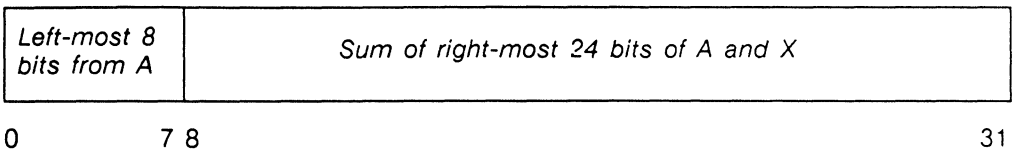
If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:

Left-most 16 bits from A	*Sum of right-most 48 bits of A and X*

0 1516 63

If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to register Z's contents according to the *usi* qualifier. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

The addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

If the comparison is not met, execution continues at the next sequential instruction.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

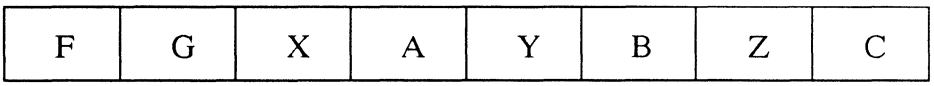
The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are specified together.

B5

Compare Integers, Branch if Greater

Full Word, Format #C
Subfunction: h000ubb0
Qualifiers : h,u=[usi],bb=[brf,brb,rel]

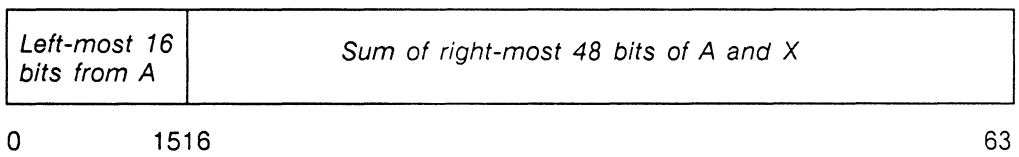


(A) + (X) GT (Z)

The #B5 instruction executes as a Compare Integer and Branch operation when bits 1 and 2 in the G designator are zero. The two operands from A and X are added, their sum compared to the integer in Z, the sum of A and X are transmitted to register C, and a branch taken according to the result.

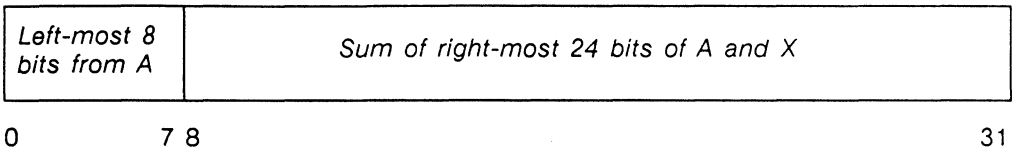
If the *h* qualifier is specified, the A, X, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to register Z's contents according to the *usi* qualifier. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

The addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

If the comparison is not met, execution continues at the next sequential instruction.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

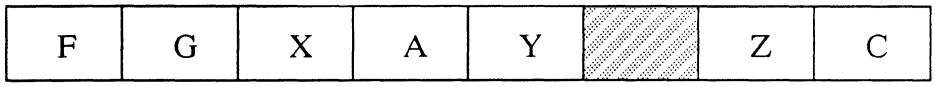
The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are specified together.

B0

Compare Integers, Set Condition if Equal

Full Word, Format #C
Subfunction: h0cfu000
Qualifiers: h,c=[sc], f=[fwc],u=[usi]

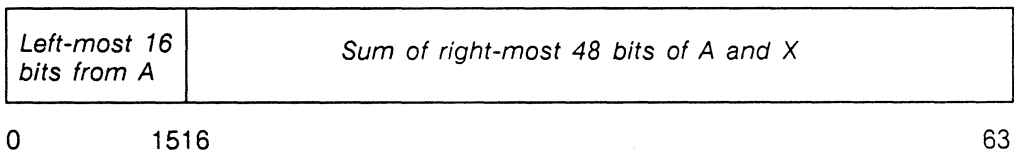


(A) + (X) EQ (Z)

The #B0 instruction executes as a Compare Integer and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The two operands from A and X are added, the sum compared to the integer in Z, the sum of A and X are transmitted to register C, and a condition code set in the register designated by Y, according to the result.

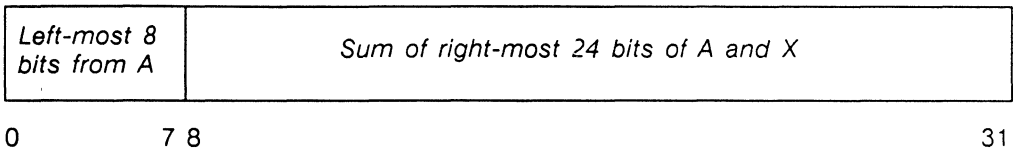
If the *h* qualifier is specified, the A, X, Y, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to the register Z's contents according to the *fwc* and *usi* qualifiers. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

If *fwc* is specified, 64 bits of the result stored to register C are compared to 64 bits previously read from register Z, otherwise the addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

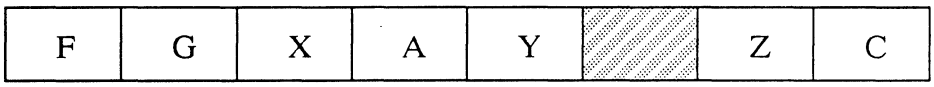
If the comparison is met, the condition code is set by loading register Y with the 64-bit (32 if *h* was specified) value 000 ... 001. If the comparison failed, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

The instruction is undefined if both *h* and *fwc* are specified, or if the C designator is equal to the Z designator.

B1

Compare Integers, Set Condition if Not Equal

Full Word, Format #C
Subfunction: h0cfu000
Qualifiers: h,c=[sc],f=[fwc],u=[usi]

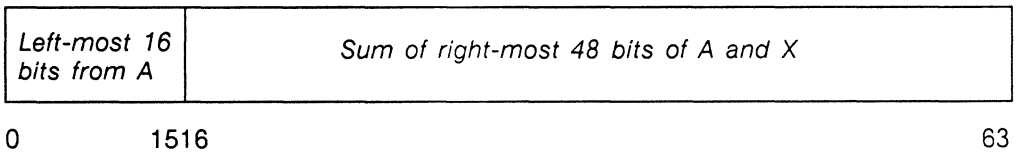


(A) + (X) NE (Z)

The #B1 instruction executes as a Compare Integer and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The two operands from A and X are added, the sum compared to the integer in Z, the sum of A and X are transmitted to register C, and a condition code set in the register designated by Y, according to the result.

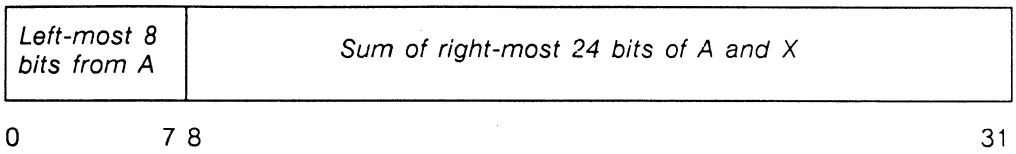
If the *h* qualifier is specified, the A, X, Y, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to the register Z's contents according to the *fwc* and *usi* qualifiers. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

If *fwc* is specified, 64 bits of the result stored to register C are compared to 64 bits previously read from register Z, otherwise the addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

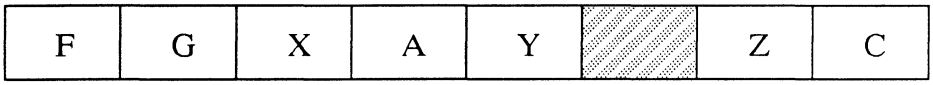
If the comparison is met, the condition code is set by loading register Y with the 64-bit (32 if *h* was specified) value 000 ... 001. If the comparison failed, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

The instruction is undefined if both *h* and *fwc* are specified, or if the C designator is equal to the Z designator.

B2

Compare Integers, Set Condition if Greater or Equal

Full Word, Format #C
Subfunction: h0c0u000
Qualifiers: h,c=[sc],u=[usi]

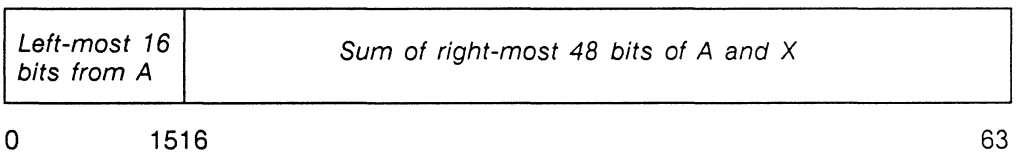


(A) + (X) GE (Z)

The #B2 instruction executes as a Compare Integer and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The two operands from A and X are added, the sum compared to the integer in Z, the sum of A and X are transmitted to register C, and a condition code set in the register designated by Y, according to the result.

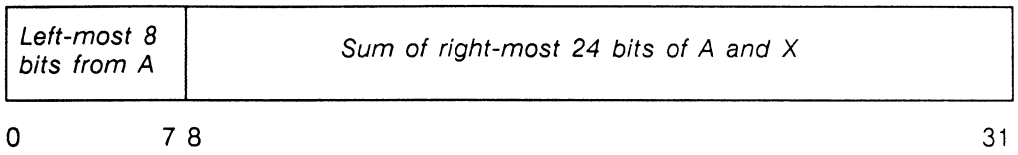
If the *h* qualifier is specified, the A, X, Y, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to the register Z's contents according to the *usi* qualifier. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

The addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

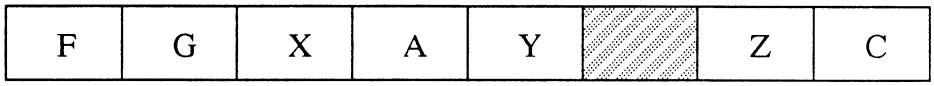
If the comparison is met, the condition code is set by loading register Y with the 64-bit (32 if *h* was specified) value 000 ... 001. If the comparison failed, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

The instruction is undefined if the C designator is equal to the Z designator.

B3

Compare Integers, Set Condition if Less

Full Word, Format #C
Subfunction: h0c0u000
Qualifiers: h,c=[sc], u=[usi]

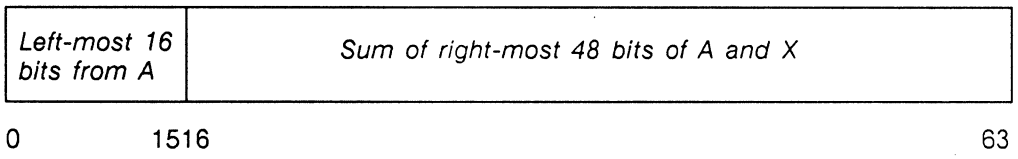


(A) + (X) LT (Z)

The #B3 instruction executes as a Compare Integer and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The two operands from A and X are added, the sum compared to the integer in Z, the sum of A and X are transmitted to register C, and a condition code set in the register designated by Y, according to the result.

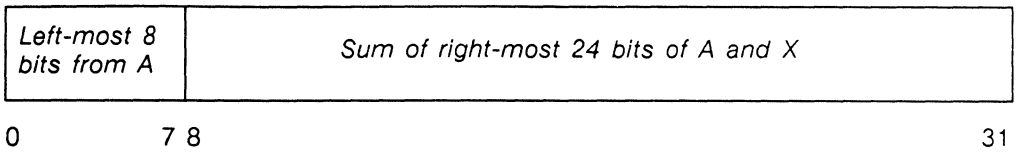
If the *h* qualifier is specified, the A, X, Y, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to the register Z's contents according to the *usi* qualifier. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

The addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

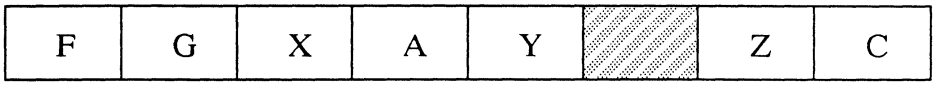
If the comparison is met, the condition code is set by loading register Y with the 64-bit (32 if *h* was specified) value 000 ... 001. If the comparison failed, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

The instruction is undefined if the C designator is equal to the Z designator.

B4

Compare Integers, Set Condition if Less or Equal

Full Word, Format #C
Subfunction: h0c0u000
Qualifiers: h,c=[sc], u=[usi]

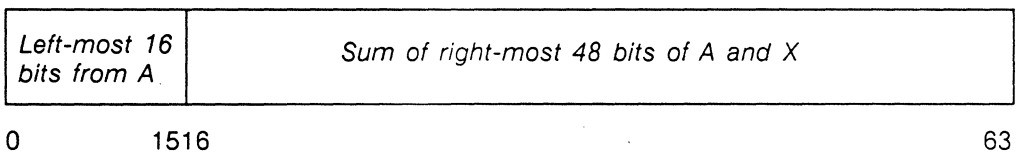


(A) + (X) LE (Z)

The #B4 instruction executes as a Compare Integer and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The two operands from A and X are added, the sum compared to the integer in Z, the sum of A and X are transmitted to register C, and a condition code set in the register designated by Y, according to the result.

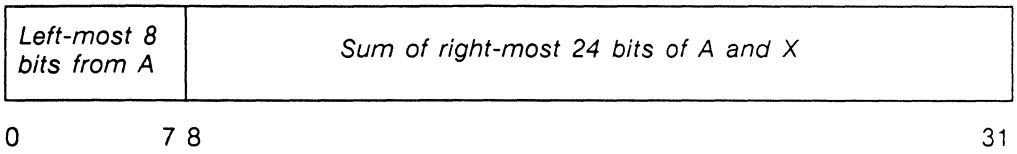
If the *h* qualifier is specified, the A, X, Y, C, and Z operands are 32-bit registers, otherwise they are 64-bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to the register Z's contents according to the *usi* qualifier. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

The addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

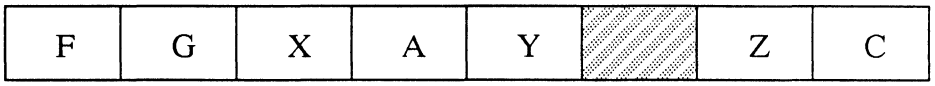
If the comparison is met, the condition code is set by loading register Y with the 64-bit (32 if *h* was specified) value 000 ... 001. If the comparison failed, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

The instruction is undefined if the C designator is equal to the Z designator.

B5

Compare Integers, Set Condition if Greater

Full Word, Format #C
Subfunction: h0c0u000
Qualifiers: h,c=[sc],u=[usi]

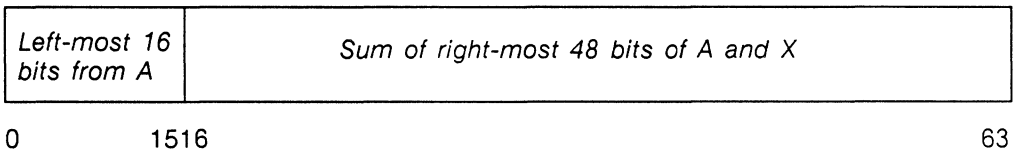


(A) + (X) GT (Z)

The #B5 instruction executes as a Compare Integer and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The two operands from A and X are added, the sum compared to the integer in Z, the sum of A and X are transmitted to register C, and a condition code set in the register designated by Y, according to the result.

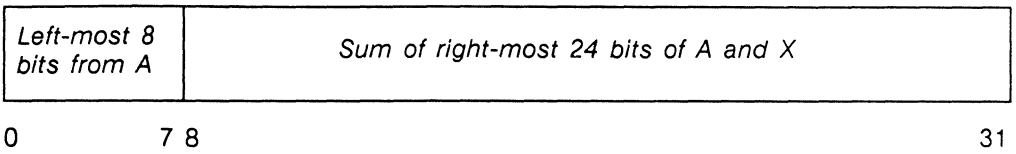
If the *h* qualifier is specified, the A, X, Y, C, and Z operands are 32–bit registers, otherwise they are 64–bit registers.

If the *h* qualifier is not specified, the integers in the right-most 48 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 48-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 16 bits of register A are transmitted to the left-most 16 bits of register C. Register C's contents are:



If the *h* qualifier specifies 32-bit operands, the integers in the right-most 24 bits of registers A and X are added, and any overflow ignored. Register C is loaded with the 24-bit result. If register #00 is specified as register A or X, machine zero is supplied. The left-most 8 bits of register A are transmitted to the left-most 8 bits of register C.

Register C's contents are:



The result is compared to the register Z's contents according to the *usi* qualifier. (If register #00 is specified for register Z, all zeros are supplied for the comparison).

The addition result is compared to the right-most 48 (or 24) bits of register Z. If *usi* is specified, the compared integers are interpreted as unsigned numbers. If not, the integers are interpreted as signed, two's complement numbers.

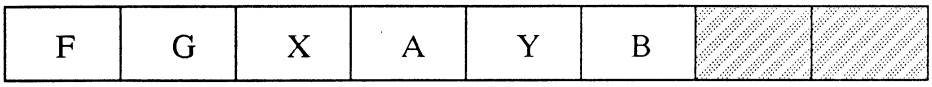
If the comparison is met, the condition code is set by loading register Y with the 64-bit (32 if *h* was specified) value 000 ... 001. If the comparison failed, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

The instruction is undefined if the C designator is equal to the Z designator.

B0

Compare Floating-Point, Branch if Equal

Full Word, Format #C
Subfunction: h1000bb0
Qualifiers: h, bb=[brf,brb,rel]



(A) EQ (X)

The #B0 instruction performs a Compare Floating-Point and Branch operation only when bit 1 of the G designator is one and bit 2 is zero. The two floating-point operands from registers A and X are compared according to the floating-point rules discussed in appendix F. If the comparison is not met, execution continues at the next sequential instruction.

If the *h* qualifier is specified, the A and X operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are specified together.

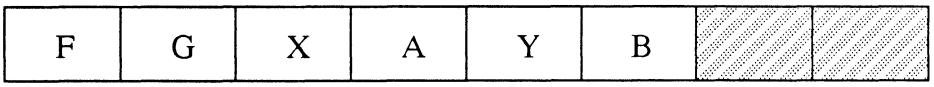
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B1

Compare Floating-Point, Branch if Not Equal

Full Word, Format #C
Subfunction: h1000bb0
Qualifiers: h, bb=[brf,brb,rel]



(A) NE (X)

The #B1 instruction performs a Compare Floating-Point and Branch operation only when bit 1 of the G designator is one and bit 2 is zero. The two floating-point operands from registers A and X are compared according to the floating-point rules discussed in appendix F. If the comparison is not met, execution continues at the next sequential instruction.

If the *h* qualifier is specified, the A and X operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are all specified together.

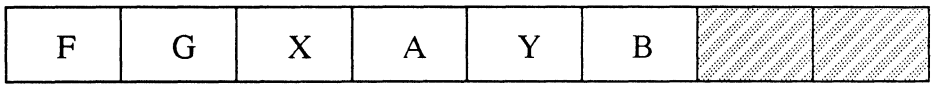
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B2

Compare Floating-Point, Branch if Greater or Equal

Full Word, Format #C
Subfunction: h1000bb0
Qualifiers: h, bb=[brf,brb,rel]



(A) GE (X)

The #B2 instruction performs a Compare Floating-Point and Branch operation only when bit 1 of the G designator is one and bit 2 is zero. The two floating-point operands from registers A and X are compared according to the floating-point rules discussed in appendix F. If the comparison is not met, execution continues at the next sequential instruction.

If the *h* qualifier is specified, the A and X operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are all specified together.

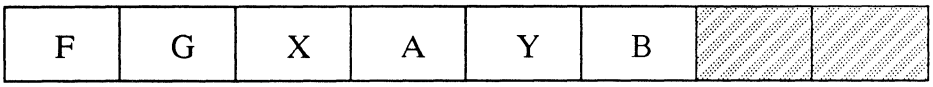
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B3

Compare Floating-Point, Branch if Less Than

Full Word, Format #C
Subfunction: h1000bb0
Qualifiers: h, bb=[brf,brb,rel]



(A) LT (X)

The #B3 instruction performs a Compare Floating-Point and Branch operation only when bit 1 of the G designator is one and bit 2 is zero. The two floating-point operands from registers A and X are compared according to the floating-point rules discussed in appendix F. If the comparison is not met, execution continues at the next sequential instruction.

If the *h* qualifier is specified, the A and X operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are all specified together.

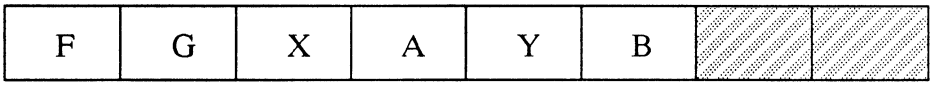
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B4

Compare Floating-Point, Branch if Less or Equal

Full Word, Format #C
Subfunction: h1000bb0
Qualifiers: h, bb=[brf,brb,rel]



(A) LE (X)

The #B4 instruction performs a Compare Floating-Point and Branch operation only when bit 1 of the G designator is one and bit 2 is zero. The two floating-point operands from registers A and X are compared according to the floating-point rules discussed in appendix F. If the comparison is not met, execution continues at the next sequential instruction.

If the *h* qualifier is specified, the A and X operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are all specified together.

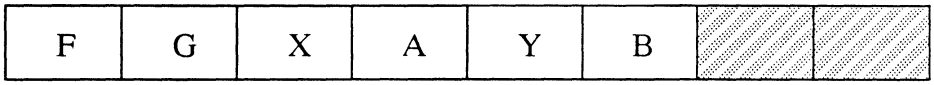
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B5

Compare Floating-Point, Branch if Greater

Full Word, Format #C
Subfunction: h1000bb0
Qualifiers: h, bb=[brf,brb,rel]



(A) GT (X)

The #B5 instruction performs a Compare Floating-Point and Branch operation only when bit 1 of the G designator is one and bit 2 is zero. The two floating-point operands from registers A and X are compared according to the floating-point rules discussed in appendix F. If the comparison is not met, execution continues at the next sequential instruction.

If the *h* qualifier is specified, the A and X operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, the instruction branches according to the specified qualifiers *brf* and *brb*. If no qualifier is specified, control branches to an address formed by adding the half word item count in 64-bit register Y, shifted left 5 places, to the base address in 64-bit register B. Otherwise, a relative branch forwards or backwards occurs.

The relative branch address is formed by taking the two 8-bit designators Y and B together as a 16-bit quantity, treated as a half word item count. This quantity is left-shifted 5 places and added to (if *brf* is specified) or subtracted from (if *brb* is specified) the instruction's program address.

The instruction is undefined if the qualifiers *brf* and *brb* are all specified together.

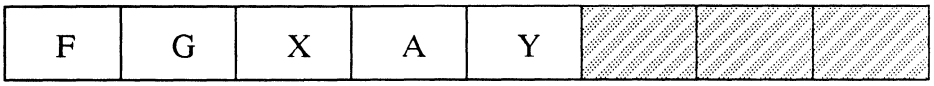
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B0

Compare Floating-Point, Set Condition if Equal

Full Word, Format #C
Subfunction: h1c00000
Qualifiers: h, c=[sc]



(A) EQ (X)

The #B0 instruction executes as a Compare Floating-Point and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The instruction compares two floating-point numbers in registers A and X according to the floating-point comparison rules explained in appendix F. A condition code is set in the register designated by Y, according to the result. If the *h* qualifier is specified, the operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, register Y is loaded with the 64-bit (32 if *h* was specified) condition code 000 ... 001. If the comparison fails, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

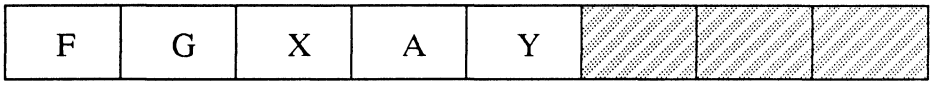
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B1

Compare Floating-Point, Set Condition if Not Equal

Full Word, Format #C
Subfunction: h1c00000
Qualifiers: h, c=[sc]



(A) NE (X)

The #B1 instruction executes as a Compare Floating-Point and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The instruction compares two floating-point numbers in registers A and X according to the floating-point comparison rules explained in appendix F. A condition code is set in the register designated by Y, according to the result. If the *h* qualifier is specified, the operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, register Y is loaded with the 64-bit (32 if *h* was specified) condition code 000 ... 001. If the comparison fails, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

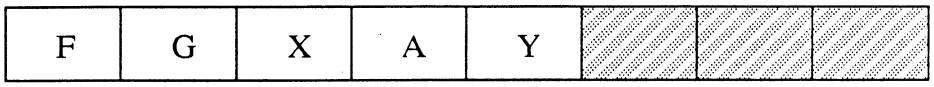
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B2

Compare Floating-Point, Set Condition if Greater or Equal

Full Word, Format #C
Subfunction: h1c00000
Qualifiers: h, c=[sc]



(A) GE (X)

The #B2 instruction executes as a Compare Floating-Point and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The instruction compares two floating-point numbers in registers A and X according to the floating-point comparison rules explained in appendix F. A condition code is set in the register designated by Y, according to the result. If the *h* qualifier is specified, the operands are 32–bit registers, otherwise they are 64–bit registers.

If the comparison is met, register Y is loaded with the 64-bit (32 if *h* was specified) condition code 000 ... 001. If the comparison fails, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

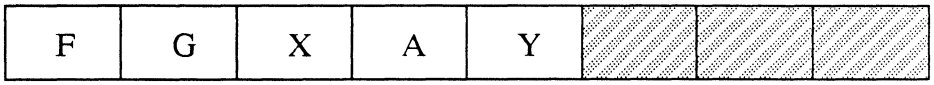
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B3

Compare Floating-Point, Set Condition if Less Than

Full Word, Format #C
Subfunction: h1c00000
Qualifiers: h, c=[sc]



(A) LT (X)

The #B3 instruction executes as a Compare Floating-Point and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The instruction compares two floating-point numbers in registers A and X according to the floating-point comparison rules explained in appendix F. A condition code is set in the register designated by Y, according to the result. If the *h* qualifier is specified, the operands are 32–bit registers, otherwise they are 64–bit registers.

If the comparison is met, register Y is loaded with the 64-bit (32 if *h* was specified) condition code 000 ... 001. If the comparison fails, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

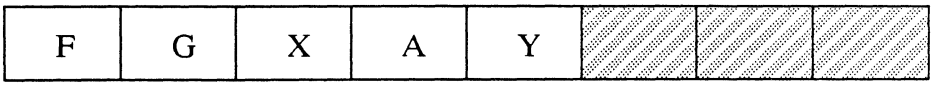
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B4

Compare Floating-Point, Set Condition if Less or Equal

Full Word, Format #C
Subfunction: h1c00000
Qualifiers: h, c=[sc]



(A) LE (X)

The #B4 instruction executes as a Compare Floating-Point and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The instruction compares two floating-point numbers in registers A and X according to the floating-point comparison rules explained in appendix F. A condition code is set in the register designated by Y, according to the result. If the *h* qualifier is specified, the operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, register Y is loaded with the 64-bit (32 if *h* was specified) condition code 000 ... 001. If the comparison fails, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

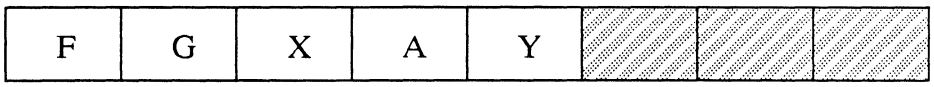
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B5

Compare Floating-Point, Set Condition if Greater

Full Word, Format #C
Subfunction: h1c00000
Qualifiers: h, c=[sc]



(A) GT (X)

The #B5 instruction executes as a Compare Floating-Point and Set Condition operation only when bit 1 of the G designator is zero, and the *sc* qualifier sets bit 2 to one. The instruction compares two floating-point numbers in registers A and X according to the floating-point comparison rules explained in appendix F. A condition code is set in the register designated by Y, according to the result. If the *h* qualifier is specified, the operands are 32-bit registers, otherwise they are 64-bit registers.

If the comparison is met, register Y is loaded with the 64-bit (32 if *h* was specified) condition code 000 ... 001. If the comparison fails, register Y is set to a condition code of 000 ... 000. Execution continues at the next sequential instruction.

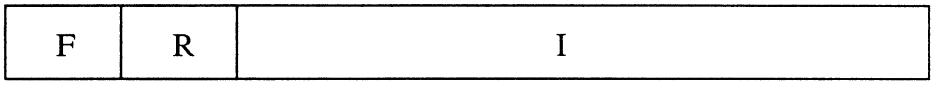
Data flag branch conditions:

Data flag bit 46: Result is indefinite

B6

Branch to Immediate Address

Full Word, Format #5
Subfunction: None



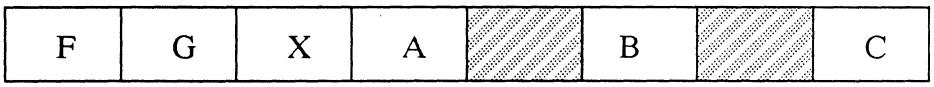
(R) + I (48 Bits)

#B6 performs an unconditional branch. The right-most 48 bits of register R contain an item count of half words, and I is a 48-bit base address. An address is formed by adding the item count, shifted left 5 places, to the base address. Overflow is ignored. If the R designator is zero, the item count to be added is all zeros.

B7

Scatter

Full Word, Format #1
Subfunction: h000bfgr
Qualifiers: h, b,f=[fia],g=[grp],r=[rf]



B ---> Indexed C

The #B7 instruction scatters groups of elements from contiguous vector B into elements of vector C. Elements of vectors B and C are 64 bits by default, or 32 bits by declaring the *h* qualifier. Elements of index vector A are 64 bits.

The locations of vector C element groups are specified by item counts contained in the right-most 48 bits of each vector A element. The first group of vector B elements 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 item count is left-shifted six places (five if *h* is specified) before the addition. The next group begins at the address formed by adding the second item count from vector A to vector C's base address, and so on, until vector A is exhausted.

Qualifier *b* indicates that register B contains a constant that is broadcast as the common value for any elements of vector B.

If the *fia* qualifier is specified, vector A is generated by using a fixed increment value specified by the right-most 48 bits of register A. The X designator must be zero. Vector C is addressed as *C, C+A, C+2A, ... , C+((N–1)A)*, where N is a field length specified by the left-most 16 bits of register A. The fixed increment value is shifted left six (or five) places before being added to vector C's base address.

If *grp* is specified, a group of elements is transmitted from vector B to vector C for each element of vector A, otherwise a single element is transmitted. The length of the group is specified in the left-most 16 bits of register C. If these bits are zero, the instruction performs as a no-op.

Qualifier *rf* indicates that all elements of vector B reside in the 256 registers of the register file (address #0–#3FCD). #B7 is undefined if *rf* is specified, but all vector B addresses are not in the register file.

The instruction is undefined if *b* is specified with *grp*, or if *grp* is specified with *rf*.

B8

Transmit Reverse

Full Word, Format #1
Subfunction: hzo00000
Qualifiers: h,z,o

A ---> C

#B8 transmits vector A's elements to vector C, in reverse order. The last element of vector A is the first vector C element, the next-to-last element of A is the second element in vector C, and so on until vector C is exhausted.

Operands are 64 bits long by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation. The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined.

BA

Gather

Full Word, Format #1
Subfunction: h0000fgr
Qualifiers : h,f=[fia],g=[grp],r=[rf]

Indexed B ---> C

The #BA instruction gathers elements from vector B and transmits them into elements of contiguous vector C. The right-most 48 bits of each element in vector A contains an item count specifying the location of each vector B element. Elements of vectors B and C are 64 bits by default, or 32 bits by declaring the *h* qualifier. Vector A elements are always 64 bits long.

The first group of vector B elements comes from an address in vector B formed by adding the first item count from vector A to the base address in register B. The item count is left-shifted six places (five if *h* is specified) before the addition. The elements are stored in vector C, in consecutive order. The operation continues until vector A is exhausted.

If the *fia* qualifier is specified, vector A is generated by using a fixed increment value specified by the right-most 48 bits of register A. The X designator must be zero. Vector B is addressed as: *B, B+A, B+2A, ... , B+((N–1)A)*, where N is a field length specified by the left-most 16 bits of register A. The fixed increment value is shifted left six places (five if *h* is specified) before being added to vector B's base address.

If *grp* is specified, a group of elements is transmitted from vector B to vector C for each element of vector A, otherwise a single element is transmitted. All groups contain the same number of elements. The length of the group is specified in the left-most 16 bits of register B. If these bits are zero, the instruction performs as a no-op.

Qualifier *rf* indicates that all elements of vector B reside in the 256 registers of the register file (address #0–#3FCD). The instruction is undefined if *rf* is specified, but all vector B addresses are not in the register file. It is also undefined if *grp* and *rf* are specified together.

BB

Mask

Full Word, Format #2
Subfunction: h00ab000
Qualifiers: h,a,b

A, B ---> C Per Z

This instruction merges elements of vectors A and B to form result vector C, as directed by the order vector Z. When a binary one is encountered in order vector Z, the next vector A element is transmitted to result vector C, and an element of vector B is skipped. When a binary zero is encountered in vector Z, the next vector B element is inserted into vector C, and an element of A skipped. Vector C's length is transmitted to bits 0-15 of register C. The #BB instruction terminates when order vector Z is exhausted.

Operands are 64 bits by default, 32 bits if the *h* qualifier is specified.

Qualifiers *a* and *b* indicate that registers A and B contain constants which are broadcast as a common value for an element of vector A and B. Either qualifier, or both, may be used.

BC

Compress

Full Word, Format #2
Subfunction: hz000000
Qualifiers: h,z

A ---> C Per Z

The #BC instruction forms the sparse data vector C by compressing vector A as directed by the order vector Z. Vector C contains elements of vector A corresponding to positions of binary ones (zeros if the *z* qualifier was specified) in the order vector Z. The length of vector C is stored into bits 0–15 of register C. Operands are 64 bits by default, 32 if the *h* qualifier is specified.

The instruction terminates when the order vector Z is exhausted.

BD

Merge

Full Word, Format #2
Subfunction: h00ab00s
Qualifiers: h,a,b,s=[sb]

A, B ---> C Per Z

The #BD instruction merges elements of vectors A and B as directed by the order vector Z. When the order vector Z contains a one in a given bit position, the next vector A element is inserted into vector C. If the vector Z bit is a zero, the next vector B element is inserted instead. No elements of vectors A or B are skipped.

If the *sb* qualifier is specified, for each vector A operand stored, the corresponding vector B element is skipped. However, a vector A element is not skipped when a vector B element is stored. The final length of vector C is stored in bits 0–15 of register C.

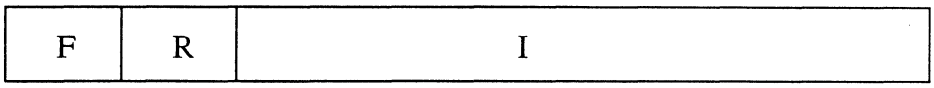
Operands are 64 bits by default, 32 bits if the *h* qualifier is specified. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vectors A and B.

The instruction terminates when the order vector Z is exhausted.

BE

Enter (R) With I (48 Bits)

Full Word, Format #5
Subfunction: None

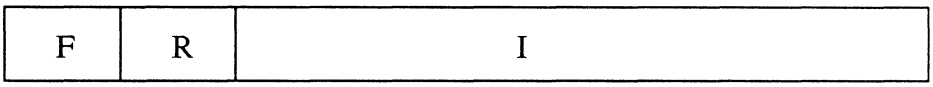


The #BE instruction transfers the 48-bit immediate operand I to the right-most 48 bits of register R, and places zeros in the upper 16 bits.

BF

Increase (R) By I (48 Bits)

Full Word, Format #5
Subfunction: None



The #BF instruction sums the right-most bits of register R and the 48-bit immediate operand, storing the result in the right-most 48 bits of register R. Arithmetic overflow is ignored. The upper 16 bits are unchanged.

C0

Select Equal; A = B, Item Count to (C)

Full Word, Format #1
Subfunction: hz0ab000
Qualifiers: h,z,a,b

F	G	X	A	Y	B	Z	C

This instruction compares each vector A element with its associated vector B element until A is equal to B, or until the shorter of the two vectors is exhausted. The comparison is performed according to the floating-point rules discussed in Appendix F.

Operands are 64 bits by default, 32 bits if the *h* qualifier is specified. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vectors A and B. If *a* or *b* is specified, the instructions terminate when the non-broadcast field terminates. The instruction is undefined if *a* and *b* are specified together.

If used, the control vector Z indicates which pairs of elements to compare. The *z* qualifier means that a zero bit in the control vector enables, and a one bit disables, the comparison for the corresponding A and B element.

An item count is stored in the right-most 48 bits of the cleared register C. The item count includes all pairs of elements encountered, not just those compared. If the comparison is met, the item count is the number of pairs of elements encountered up to, but not including, the pair meeting the condition. If vectors A and B are exhausted before a permissive control vector element is encountered, the item count equals the shorter vector's length (determined after the offset adjustment).

If the C designator is zero, results are undefined.

Data flag branch conditions:

Data flag bit 37:	The select condition was not met.
Data flag bit 46:	Indefinite result.

C1

Select Not Equal; A NE B, Item Count to (C)

Full Word, Format #1
Subfunction: hz0ab000
Qualifiers: h,z,a,b

F	G	X	A	Y	B	Z	C

This instruction compares each vector A element with its associated vector B element until A is not equal to B, or until the shorter of the two vectors is exhausted. The comparison is performed according to the floating-point rules discussed in Appendix F.

Operands are 64 bits by default, 32 bits if the *h* qualifier is specified. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vectors A and B. If *a* or *b* is specified, the instructions terminate when the non-broadcast field terminates. The instruction is undefined if *a* and *b* are specified together.

If used, the control vector Z indicates which pairs of elements to compare. The *z* qualifier means that a zero bit in the control vector enables, and a one bit disables, the comparison for the corresponding A and B element.

An item count is stored in the right-most 48 bits of the cleared register C. The item count includes all pairs of elements encountered, not just those compared. If the comparison is met, the item count is the number of pairs of elements encountered up to, but not including, the pair meeting the condition. If vectors A and B are exhausted before a permissive control vector element is encountered, the item count equals the shorter vector's length (determined after the offset adjustment).

If the C designator is zero, results are undefined.

Data flag branch conditions:

Data flag bit 37:	The select condition was not met.
Data flag bit 46:	Indefinite result.

C2

Select Greater or Equal; A GE B, Item Count to (C)

Full Word, Format #1
Subfunction: hz0ab000
Qualifiers: h,z,a,b

F	G	X	A	Y	B	Z	C

This instruction compares each vector A element with its associated vector B element until A is greater than or equal to B, or until the shorter of the two vectors is exhausted. The comparison is performed according to the floating-point rules discussed in Appendix F.

Operands are 64 bits by default, 32 bits if the *h* qualifier is specified. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vectors A and B. If *a* or *b* is specified, the instructions terminate when the non-broadcast field terminates. The instruction is undefined if *a* and *b* are specified together.

If used, the control vector Z indicates which pairs of elements to compare. The *z* qualifier means that a zero bit in the control vector enables, and a one bit disables, the comparison for the corresponding A and B element.

An item count is stored in the right-most 48 bits of the cleared register C. The item count includes all pairs of elements encountered, not just those compared. If the comparison is met, the item count is the number of pairs of elements encountered up to, but not including, the pair meeting the condition. If vectors A and B are exhausted before a permissive control vector element is encountered, the item count equals the shorter vector's length (determined after the offset adjustment).

If the C designator is zero, results are undefined.

Data flag branch conditions:

Data flag bit 37:	The select condition was not met.
Data flag bit 46:	Indefinite result.

C3

Select Less; A LT B, Item Count to (C)

Full Word, Format #1
Subfunction: hz0ab000
Qualifiers: h,z,a,b

F	G	X	A	Y	B	Z	C

This instruction compares each vector A element with its associated vector B element until A is less than B, or until the shorter of the two vectors is exhausted. The comparison is performed according to the floating-point rules discussed in Appendix F.

Operands are 64 bits by default, 32 bits if the *h* qualifier is specified. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vectors A and B. If *a* or *b* is specified, the instructions terminate when the non-broadcast field terminates. The instruction is undefined if *a* and *b* are specified together.

If used, the control vector Z indicates which pairs of elements to compare. The *z* qualifier means that a zero bit in the control vector enables, and a one bit disables, the comparison for the corresponding A and B element.

An item count is stored in the right-most 48 bits of the cleared register C. The item count includes all pairs of elements encountered, not just those compared. If the comparison is met, the item count is the number of pairs of elements encountered up to, but not including, the pair meeting the condition. If vectors A and B are exhausted before a permissive control vector element is encountered, the item count equals the shorter vector's length (determined after the offset adjustment).

If the C designator is zero, results are undefined.

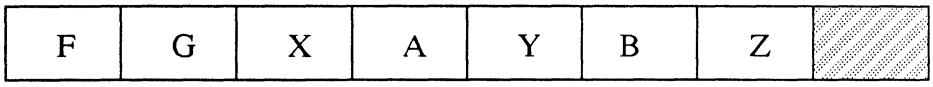
Data flag branch conditions:

Data flag bit 37:	The select condition was not met.
Data flag bit 46:	Indefinite result.

C4

Compare; Equal

Full Word, Format #1
Subfunction: h00ab000
Qualifiers: h,a,b



A EQ B Order Vector ---> Z

The #C4 instruction compares successive elements of vector A to successive elements of vector B according to floating-point comparison rules (described in Appendix F). If the comparison is met, the corresponding bit of order vector Z is set, otherwise it is cleared to zero. The instruction terminates when vector Z is filled. Elements of vectors A and B may be 64 bits by default, or 32 bits by declaring the *h* qualifier.

Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vector A and B. Either *a*, *b*, or both may be used.

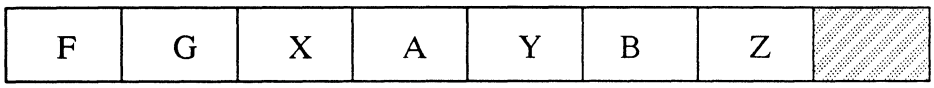
Data flag branch conditions:

Data flag bit 46: Indefinite result.

C5

Compare; Not Equal

Full Word, Format #1
Subfunction: h00ab000
Qualifiers: h,a,b



A NE B Order Vector ---> Z

The #C5 instruction compares successive elements of vector A to successive elements of vector B according to floating-point comparison rules (described in Appendix F). If the comparison is met, the corresponding bit of order vector Z is set, otherwise it is cleared to zero. The instruction terminates when vector Z is filled. Elements of vectors A and B may be 64 bits by default, or 32 bits by declaring the *h* qualifier.

Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vector A and B. Either *a*, *b*, or both may be used.

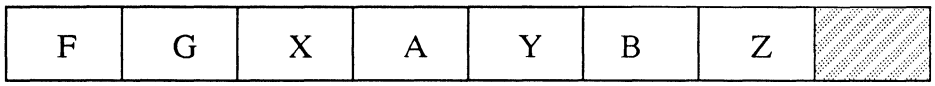
Data flag branch conditions:

Data flag bit 46: Indefinite result.

C6

Compare; Greater Than or Equal

Full Word, Format #1
Subfunction: h00ab000
Qualifiers: h,a,b



A GE B Order Vector ---> Z

The #C6 instruction compares successive elements of vector A to successive elements of vector B according to floating-point comparison rules (described in Appendix F). If the comparison is met, the corresponding bit of order vector Z is set, otherwise it is cleared to zero. The instruction terminates when vector Z is filled. Elements of vectors A and B may be 64 bits by default, or 32 bits by declaring the *h* qualifier.

Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vector A and B. Either *a*, *b*, or both may be used.

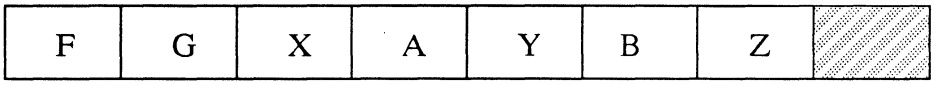
Data flag branch conditions:

Data flag bit 46: Indefinite result.

C7

Compare; Less

Full Word, Format #1
Subfunction: h00ab000
Qualifiers: h,a,b



A LT B Order Vector ---> Z

The #C7 instruction compares successive elements of vector A to successive elements of vector B according to floating-point comparison rules (described in Appendix F). If the comparison is met, the corresponding bit of order vector Z is set, otherwise it is cleared to zero. The instruction terminates when vector Z is filled. Elements of vectors A and B may be 64 bits by default, or 32 bits by declaring the *h* qualifier.

Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vector A and B. Either *a*, *b*, or both may be used.

The X and Y designators contain offsets for vectors A and B. When a constant is broadcast for either vector, that vector has no length, and the offset is ignored.

Data flag branch conditions:

Data flag bit 46: Indefinite result.

C8

Search for Equality

Full Word, Format #1
Subfunction: hzl00000
Qualifiers: h,z,l=[lh]

Search EQ; Index List ---> C

The #C8 instruction performs a search and compare operation for each element of vector A against successive elements of vector B, according to floating-point comparison rules (described in Appendix F). Each search iteration for a vector A element begins with the first element of the B field and terminates when the comparison is met, or when vector B is exhausted. After each search iteration, the element in vector C is cleared, then loaded with the index of the vector B element that caused the search to terminate (or the B field length). The resulting index is a 64-bit word with the index in the right-most 48 bits, and the left-most 16 bits cleared to zero. This index, shifted and added to the address of the first element in vector B will form the address of the vector B element that met the comparison. (A compare on the first element of vector B results in an index of zero.) The instruction terminates when vector A is exhausted.

If the *lh* qualifier is specified, each successive search starts at the location of the last successful hit in vector B (or end of B field if no hit). If *lh* is not specified, the search starts at the beginning of vector B for each vector A element.

Elements of vectors A and B are 64 bits by default, or 32 bits by declaring the *h* qualifier. Register Z may specify a control vector, each bit of which is associated with a single vector C element, that controls which elements will store results from this operation (and set data flag bit 46). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid.

Data flag branch conditions:

Data flag bit 46: Indefinite result.

C9

Search for Inequality

Full Word, Format #1
Subfunction: hzl00000
Qualifiers: h,z,l=[lh]

Search NE; Index List ---> C

The #C9 instruction performs a search and compare operation for each element of vector A against successive elements of vector B, according to floating-point comparison rules (described in Appendix F). Each search iteration for a vector A element begins with the first element of the B field and terminates when the comparison is met, or when vector B is exhausted. After each search iteration, the element in vector C is cleared, then loaded with the index of the vector B element that caused the search to terminate (or the B field length). The resulting index is a 64-bit word with the index in the right-most 48 bits, and the left-most 16 bits cleared to zero. This index, shifted and added to the address of the first element in vector B will form the address of the vector B element that met the comparison. (A compare on the first element of vector B results in an index of zero.) The instruction terminates when vector A is exhausted.

If the *lh* qualifier is specified, each successive search starts at the location of the last successful hit in vector B (or end of B field if no hit). If *lh* is not specified, the search starts at the beginning of vector B for each vector A element.

Elements of vectors A and B are 64 bits by default, or 32 bits by declaring the *h* qualifier. Register Z may specify a control vector, each bit of which is associated with a single vector C element, that controls which elements will store results from this operation (and set data flag bit 46). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid.

Data flag branch conditions:

Data flag bit 46: Indefinite result.

CA

Search for Greater

Full Word, Format #1
Subfunction: hzl00000
Qualifiers: h,z,l=[lh]

Search GE; Index List ---> C

The #CA instruction performs a search and compare operation for each element of vector A against successive elements of vector B, according to floating-point comparison rules (described in Appendix F). Each search iteration for a vector A element begins with the first element of the B field and terminates when the comparison is met, or when vector B is exhausted. After each search iteration, the element in vector C is cleared, then loaded with the index of the vector B element that caused the search to terminate (or the B field length). The resulting index is a 64-bit word with the index in the right-most 48 bits, and the left-most 16 bits cleared to zero. This index, shifted and added to the address of the first element in vector B will form the address of the vector B element that met the comparison. (A compare on the first element of vector B results in an index of zero.) The instruction terminates when vector A is exhausted.

If the *lh* qualifier is specified, each successive search starts at the location of the last successful hit in vector B (or end of B field if no hit). If *lh* is not specified, the search starts at the beginning of vector B for each vector A element.

Elements of vectors A and B are 64 bits by default, or 32 bits by declaring the *h* qualifier. Register Z may specify a control vector, each bit of which is associated with a single vector C element, that controls which elements will store results from this operation (and set data flag bit 46). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid.

Data flag branch conditions:

Data flag bit 46: Indefinite result.

CB

Search for Less

Full Word, Format #1
Subfunction: hzl00000
Qualifiers: h,z,l=[lh]

Search LT; Index List ---> C

The #CB instruction performs a search and compare operation for each element of vector A against successive elements of vector B, according to floating-point comparison rules (described in Appendix F). Each search iteration for a vector A element begins with the first element of the B field and terminates when the comparison is met, or when vector B is exhausted. After each search iteration, the element in vector C is cleared, then loaded with the index of the vector B element that caused the search to terminate (or the B field length). The resulting index is a 64-bit word with the index in the right-most 48 bits, and the left-most 16 bits cleared to zero. This index, shifted and added to the address of the first element in vector B will form the address of the vector B element that met the comparison. (A compare on the first element of vector B results in an index of zero.) The instruction terminates when vector A is exhausted.

If the *lh* qualifier is specified, each successive search starts at the location of the last successful hit in vector B (or end of B field if no hit). If *lh* is not specified, the search starts at the beginning of vector B for each vector A element.

Elements of vectors A and B are 64 bits by default, or 32 bits by declaring the *h* qualifier. Register Z may specify a control vector, each bit of which is associated with a single vector C element, that controls which elements will store results from this operation (and set data flag bit 46). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid.

Data flag branch conditions:

Data flag bit 46: Indefinite result.

CC

Masked Binary Compare

Full Word, Format #D
Subfunction: 0000000n
Qualifiers: n=[neq]

A EQ/NE (B) per (C)

The #CC instruction searches field A for a match with the contents of the 64-bit register specified by designator B. Each element of field A is logically ANDed with the 64-bit register C contents, then compared to the logical AND of registers B and C, until a match is found. Register C contains the mask word; a zero bit on the C word causes a compare EQ on that bit position.

The match is made when A equals B, unless the *neq* qualifier is specified. In this case, the match is made on inequality.

Register X is the index into field A. X is incremented by one for each word search that does not find a match. When a match is found, the index provides a means of locating the word in field A that matches register B's contents.

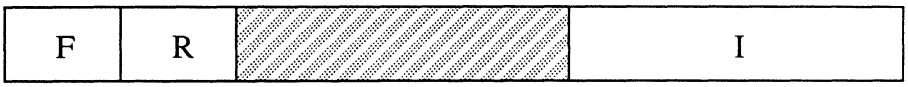
Data flag branch conditions:

Data flag bit 37: Set if no match was made.

CD

Half Word Enter (R) by I (24 Bits)

Full Word, Format #5
Subfunction: None

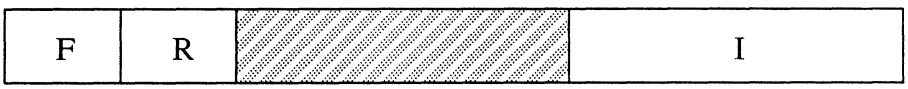


This instruction transfers the 24-bit immediate operand to the right-most 24 bits of 32-bit register R, and places zeros in the upper 8 bits of R.

CE

Half Word Increase (R) by I (24 Bits)

Full Word, Format #5
Subfunction: None

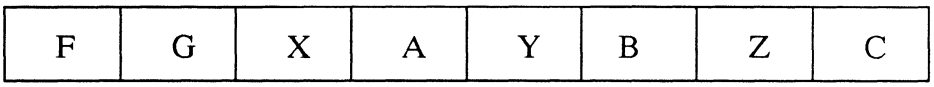


This instruction sums the right-most bits of the 32-bit register R and the 24-bit immediate operand, storing the result in the right-most 24 bits of register R. Arithmetic overflow is ignored.

CF

Arithmetic Compress

Full Word, Format #1
Subfunction: h000bsss
Qualifiers: h,b,sss=[ma,c,(n=ma+c),mb]



A ---> C per B

Arithmetic compress performs a floating-point comparison of elements of vectors A and B, forming the sparse data vector C and associated sparse order vector Z. Elements of vectors A, B and C are 64 bits by default, or 32 bits by declaring the *h* qualifier. Each vector A element that is greater than or equal to the corresponding element of vector B becomes a vector C element, and the corresponding bit of vector Z is set to one. When vector B's length is exhausted, it is extended with machine zero for the comparison,

The operation terminates when vector A is exhausted. The number of operations performed (the bit length of order vector Z) is stored in bits 0–15 of register Z, and the number of operands copied into sparse data vector C is stored into bits 0–15 of register C. If Z and C are the same, Z and C results are undefined.

If a vector A element is less than the associated element of vector B, no element is stored (or skipped) in vector C. The associated order vector Z bit is cleared to zero.

Qualifier *b* indicates that register B contains the B source field constant that is broadcast as the common value for any elements of vector B.

The qualifiers that control the state of the sign control subfunction bits are discussed in chapter 2. Although sign control qualifiers may specify operations on elements of vectors A and B before the comparison, if an element of A is stored in vector C, it is the original element.

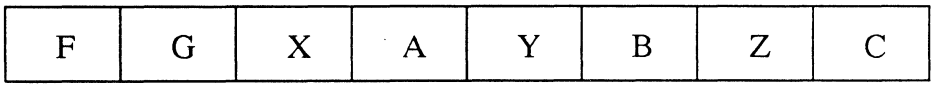
Data flag branch conditions:

Data flag bit 46: Indefinite result.

D0

Average

Full Word, Format #1
Subfunction: hzoab000
Qualifiers: h,z,o,a,b



(A(N) + B(N))/2 ---> C(N)

#D0 forms the normalized average of elements in vectors A and B by summing corresponding A and B elements, and dividing the result by two. The result becomes the corresponding element in vector C. Division is accomplished by reducing the sum's exponent by one.

Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zero bits instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers designated by C and (C+1) are undefined. Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vectors A and B.

Data flag branch conditions:

Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

D1

Adjacent Mean

Full Word, Format #1
Subfunction: hzo00000
Qualifiers: h,z,o

(A(N+1) + A(N))/2 ---> C(N)

The #D1 instruction performs a normalized addition of the *n*th and *n*th+1 elements of vector A, and divides the result by two. The final result is stored in the *n*th element of vector C. Division is accomplished by subtracting one from the sum's exponent.

Elements of vectors A and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zeros instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers C and (C+1) are undefined.

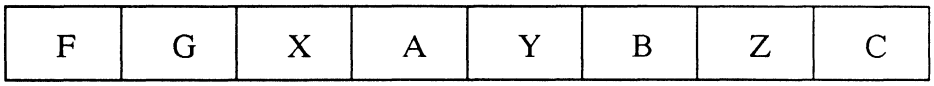
Data flag branch conditions:

Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

D4

Average Difference

Full Word, Format #1
Subfunction: hzoab000
Qualifiers: h,z,o,a,b



(A(N)–B(N))/2 ---> C(N)

The #D4 instruction takes the normalized difference of the *n*th elements of vectors A and B, and divides it by two. The result becomes the corresponding vector C element. Division is accomplished by subtracting one from the difference's exponent.

Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zeros instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers C and (C+1) are undefined.

Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vectors A and B.

Data flag branch conditions:

Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

D5

Delta

Full Word, Format #1
Subfunction: hzo00000
Qualifiers: h,z,o

(A(N+1)-A(N)) ---> C(N)

The #D5 instruction subtracts the *n*th element of vector A from the *n*th+1 element of vector A, and stores the final result in the *n*th element of vector C. Normalized floating-point arithmetic is used in the subtraction.

Elements of vectors A and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will store results from this operation (and set data flag bits). The *z* qualifier causes the control vector to operate on zeros instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid. The qualifier *o* specifies an offset for result vector C and control vector Z. The offset is found in register (C+1). Register C must be even if *o* is declared, otherwise references to registers C and (C+1) are undefined.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	Result is machine zero.
Data flag bit 46:	Indefinite result.

D8

Maximum of Vector A

Full Word, Format #1
Subfunction: hz000s00
Qualifiers: h,z,s=[ma]

Maximum of A to (C), Item Count to (B)

The #D8 instruction searches and compares successive elements of vector A for the maximum element, and moves that maximum element to register C. The number of elements up to, but not including, the located element, is stored as an item count in the right-most 48 bits of the cleared register B. If more than one element meets the comparison, data flag 54 is set, and the item count and element stored is for the first such element. The instruction terminates when vector A is exhausted. If the *h* qualifier is specified, the A operands and register C are 32 bits long, otherwise the default length of 64 bits applies.

Register Z may specify a control vector, each bit of which is associated with a single vector A element, controlling the elements that are examined for this operation (and setting data flag bits). The *z* qualifier causes a vector A element to be examined on binary zeros in the control vector, instead of binary ones. No control vector offset or length is defined. A zero Z designator causes all elements to be included, and the *z* qualifier is ignored. If the control vector has no permissive elements, no vector A elements are examined, and C's contents are undefined. The item count in register B is equal to vector A's length minus its offset.

Sign control is valid using the *ma* qualifier, which compares the magnitude of vector A's elements. The unaltered element, as read from vector A, is stored in vector C.

If the B and C designators are the same, results in B and C are undefined. If an indefinite element is encountered, data flag 46 is set, and register C set to indefinite. The contents of register B and data flag 54 are then undefined.

Data flag branch conditions:

Data flag bit 46:	Indefinite result.
Data flag bit 54:	More than one quantity met the criteria for maximum.

D9

Minimum of Vector A

Full Word, Format #1
Subfunction: hz000s00
Qualifiers: h,z,s=[ma]

Minimum of A to (C), Item Count to (B)

The #D9 instruction searches and compares successive elements of vector A for the minimum element, and moves the minimum element to register C. The number of elements up to, but not including, the located element, is stored as an item count in the right-most 48 bits of the cleared register B. If more than one element meets the comparison, data flag 54 is set, and the item count and element stored is for the first such element. The instruction terminates when vector A is exhausted. If the *h* qualifier is specified, the A operands and register C are 32 bits long, otherwise the default length of 64 bits applies.

Register Z may specify a control vector, each bit of which is associated with a single vector A element, controlling the elements that are examined for this operation (and the setting of data flag bits). The *z* qualifier causes a vector A element to be examined on binary zeros in the control vector, instead of binary ones. No control vector offset or length is defined. A zero Z designator causes all elements to be included, and the *z* qualifier is ignored. If the control vector has no permissive elements, no vector A elements are examined, and C's contents are undefined. The item count in register B is equal to vector A's length minus its offset.

Sign control is valid using the *ma* qualifier, which compares the magnitude of vector A's elements. The unaltered element, as read from vector A, is stored in vector C.

If the B and C designators are the same, results in B and C are undefined. If an indefinite element is encountered, data flag 46 is set, and register C set to indefinite. The contents of register B and data flag 54 are then undefined.

Data flag branch conditions:

Data flag bit 46:	Indefinite result.
Data flag bit 54:	More than one quantity met the criteria for minimum.

DA

Sum Vector A Elements

Full Word, Format #1
Subfunction: hz000000
Qualifiers: h,z

(A0+A1+A2+ ...+An) to (C) and (C+1)

This instruction sums all the elements in vector A, performing a double-precision floating-point operation without normalization. The upper result is stored in the register designated as C, and the lower result in C+1. Registers C and C+1 are 64 bits by default, 32 bits if the *h* qualifier is specified. Register C must be even. If register C is an odd number, or zero, results are undefined. The instruction terminates when vector A is exhausted. The final result may depend on the order of the input operands.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, controlling which elements will be summed in this operation. The *z* qualifier causes the control vector to operate on zeros instead of ones. No control vector offset or length is defined. If the control vector has no permissive elements, no vector A elements are examined, the result is machine zero, and data flag 43 is set. A zero Z designator causes all elements to be included, and the *z* qualifier is ignored.

Data flag 43 is determined by the final result only. It is set if the lower result is machine zero, regardless of the upper result. If the upper result is indefinite, the lower result is undefined. Data flags 42 and 46 will be set normally, as required on any of the add operations.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	The lower result is machine zero.
Data flag bit 46:	Indefinite result.

DB

Product of Vector A Elements

Full Word, Format #1
Subfunction: hz000000
Qualifiers: h,z

(A0*A1*A2*A3 ...*An) to (C)

The #DB instruction forms the significant product of successive floating-point elements in vector A, storing the result in register C. Register C is 64 bits long, or 32 bits, if the *h* qualifier is specified. The number of significant bits in the partial product is adjusted after each multiplication. The instruction terminates when vector A is exhausted. The final result may depend on the order of the input operands.

Register Z may specify a control vector, each bit of which is associated with a single vector A element, controlling which elements will be multiplied for this operation. Multiplication of a vector A element and a partial product takes place only when the corresponding control vector bit is enabled as specified by the *z* qualifier. If the control vector contains no permissive elements, the result is a normalized one. A zero Z designator causes all elements to be included, and the *z* qualifier is ignored.

If the C designator is zero, the result is undefined.

Data flags 43 and 46 are determined only by the final result. Data flag 42 is set if any multiplication operation overflows.

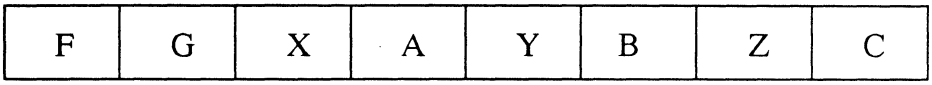
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	The result is machine zero.
Data flag bit 46:	Indefinite result.

DC

Dot Product of Vectors A and B

Full Word, Format #1
Subfunction: hz0ab000
Qualifiers: h,z,a,b



Dot Product to (C) and (C+1)

The #DC instruction multiplies vector A by vector B and forms the sum of the products, using double-precision, unnormalized arithmetic. The upper and lower results are stored in registers C and C+1 respectively. The instruction terminates when the shorter source vector is exhausted. The final result may depend on the order of the input operands.

Elements of vectors A, B, and C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector A and B element, controlling which elements will be included in this operation. The *z* qualifier causes the control vector to operate on zeros instead of ones. If register Z is zero, there is no control vector, all elements are included, and the *z* qualifier is invalid. If the control vector has no permissive elements, the result is machine zero, and data flag 43 is set.

Qualifiers *a* and *b* indicate that registers A and B contain constants that are broadcast as the common value for any elements of vectors A and B. If both *a* and *b* qualifiers are specified, the instruction is undefined.

Data flags 43 and 46 are determined only by the final upper and lower results. If the upper result is indefinite, the lower result is undefined. Data flag 43 is set if the lower result is machine zero, regardless of the upper result. Data flag 42 is set if any multiplication or addition operation overflows.

Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	The result is machine zero.
Data flag bit 46:	Indefinite result.

DF

Interval

Full Word, Format #1
Subfunction: hzo00000
Qualifiers: h,z,o

(A) per (B) ---> C

This instruction forms a result vector C whose initial element is the constant from register A. Each succeeding vector C element exceeds the preceding element by the constant in register B. The second element equals the first element of C, plus the contents of B, and so on. Arithmetic is unnormalized.

Elements of registers A and B and vector C are 64 bits by default, or 32 bits by declaring the *h* qualifier.

Register Z may specify a control vector, each bit of which is associated with a single vector C element, that controls which elements will store results form this operation (and set data flag bits).

The *z* qualifier causes the control vector to operate on zeros instead of ones. If the Z designator is zero, there is no control vector, all results are stored, and the *z* qualifier is invalid.

For each non-permissive bit in the control vector Z, the addition operation is performed, but the result is not stored in vector C. If the result of this addition is indefinite, data flag 46 is not set until a permissive bit is encountered in the control vector, so a result can be stored. Similarly, data flag bits 42 or 43 are set on the next permitted store, although the step that caused the flag to be set was not stored.

If the A designator is zero, then #8000 ... 0 is supplied for the value of A.

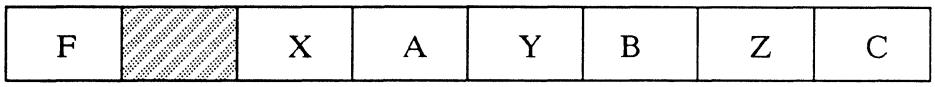
Data flag branch conditions:

Data flag bit 42:	Exponent overflow.
Data flag bit 43:	The result is machine zero.
Data flag bit 46:	Indefinite result.

F0

Logical Exclusive OR

Full Word, Format #3
Subfunction: none



A Excl. OR B ---> C

The #F0 instruction performs a bit-by-bit *logical exclusive OR* function on binary fields A and B. The result is stored in field C. The operation's results, based on bit settings of A and B, are listed below.

Source A B	Result C
0 0	0
0 1	1
1 0	1
1 1	0

The binary fields A, B, and C are strings of bits. The operation proceeds from left to right, terminating when the C field is exhausted. Item counts are bit counts.

If fields A and B are shorter than field C, they are extended automatically with binary zeros. Registers X, Y, and Z contain bit indexes that are added to the A, B, and C addresses, respectively.

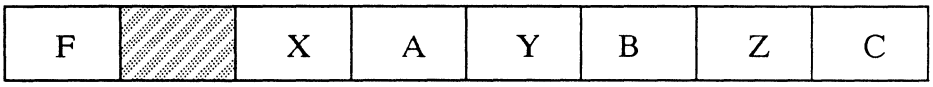
Data Flag Branch Conditions:

Data flag bit 53 — Result field all zeros
Data flag bit 54 — Result field mixed
Data flag bit 55 — Result field all ones

F1

Logical AND

Full Word, Format #3
Subfunction: none



A AND B ---> C

The #F1 instruction performs a bit-by-bit *logical AND* function on binary fields A and B. The result is stored in field C. The operation's results, based on bit settings of A and B, are listed below.

Source A B	Result C
0 0	0
0 1	0
1 0	0
1 1	1

The binary fields A, B, and C are strings of bits. The operation proceeds from left to right, terminating when the C field is exhausted. Item counts are bit counts.

If fields A and B are shorter than field C, they are extended automatically with binary zeros. Registers X, Y, and Z contain bit indexes that are added to the A, B, and C addresses, respectively.

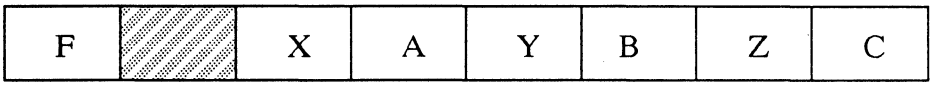
Data Flag Branch Conditions:

Data flag bit 53 — Result field all zeros
Data flag bit 54 — Result field mixed
Data flag bit 55 — Result field all ones

F2

Logical Inclusive OR

Full Word, Format #3
Subfunction: none



A OR B ---> C

The #F2 instruction performs a bit-by-bit logical inclusive OR function on binary fields A and B. The result is stored in field C. The operation's results, based on bit settings of A and B, are listed below.

Source A B	Result C
0 0	0
0 1	1
1 0	1
1 1	1

The binary fields A, B, and C are strings of bits. The operation proceeds from left to right, terminating when the C field is exhausted. Item counts are bit counts.

If fields A and B are shorter than field C, they are extended automatically with binary zeros. Registers X, Y, and Z contain bit indexes that are added to the A, B, and C addresses, respectively.

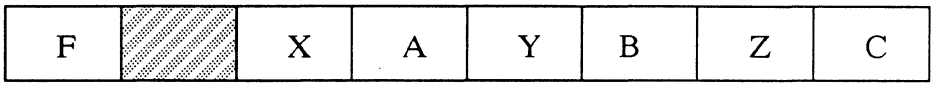
Data Flag Branch Conditions:

Data flag bit 53 — Result field all zeros
Data flag bit 54 — Result field mixed
Data flag bit 55 — Result field all ones

F3

Logical NOT AND

Full Word, Format #3
Subfunction: none



NOT of A AND B ---> C

The #F3 instruction performs a bit-by-bit logical NOT AND function on binary fields A and B. The result is stored in field C. The operation's results, based on bit settings of A and B, are listed below.

Source A B	Result C
0 0	1
0 1	1
1 0	1
1 1	0

The binary fields A, B, and C are strings of bits. The operation proceeds from left to right, terminating when the C field is exhausted. Item counts are bit counts.

If fields A and B are shorter than field C, they are extended automatically with binary zeros. Registers X, Y, and Z contain bit indexes that are added to the A, B, and C addresses, respectively.

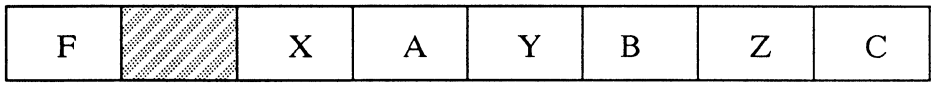
Data Flag Branch Conditions:

Data flag bit 53 — Result field all zeros
Data flag bit 54 — Result field mixed
Data flag bit 55 — Result field all ones

F4

Logical NOT OR

Full Word, Format #3
Subfunction: none



NOT of A OR B ---> C

The #F4 instruction performs a bit-by-bit logical NOT OR function on binary fields A and B. The result is stored in field C. The operation's results, based on bit settings of A and B, are listed below.

Source A B	Result C
0 0	1
0 1	0
1 0	0
1 1	0

The binary fields A, B, and C are strings of bits. The operation proceeds from left to right, terminating when the C field is exhausted. Item counts are bit counts.

If fields A and B are shorter than field C, they are extended automatically with binary zeros. Registers X, Y, and Z contain bit indexes that are added to the A, B, and C addresses, respectively.

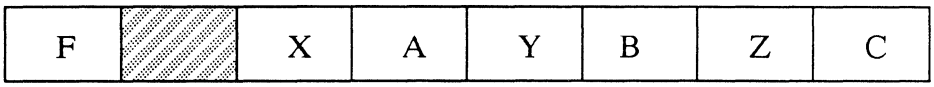
Data Flag Branch Conditions:

Data flag bit 53 — Result field all zeros
Data flag bit 54 — Result field mixed
Data flag bit 55 — Result field all ones

F5

Logical OR NOT

Full Word, Format #3
Subfunction: none



A OR NOT B ---> C

The #F5 instruction performs a bit-by-bit logical OR NOT function on binary fields A and B. The result is stored in field C. The operation's results, based on bit settings of A and B, are listed below.

Source A B	Result C
0 0	1
0 1	0
1 0	1
1 1	1

The binary fields A, B, and C are strings of bits. The operation proceeds from left to right, terminating when the C field is exhausted. Item counts are bit counts.

If fields A and B are shorter than field C, they are extended automatically with binary zeros. Registers X, Y, and Z contain bit indexes that are added to the A, B, and C addresses, respectively.

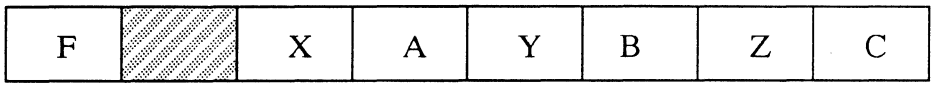
Data Flag Branch Conditions:

Data flag bit 53 — Result field all zeros
Data flag bit 54 — Result field mixed
Data flag bit 55 — Result field all ones

F6

Logical AND NOT

Full Word, Format #3
Subfunction: none



A AND NOT B ---> C

The #F6 instruction performs a bit-by-bit logical AND NOT function on binary fields A and B. The result is stored in field C. The operation's results, based on bit settings of A and B, are listed below.

Source A B	Result C
0 0	0
0 1	0
1 0	1
1 1	0

The binary fields A, B, and C are strings of bits. The operation proceeds from left to right, terminating when the C field is exhausted. Item counts are bit counts.

If fields A and B are shorter than field C, they are extended automatically with binary zeros. Registers X, Y, and Z contain bit indexes that are added to the A, B, and C addresses, respectively.

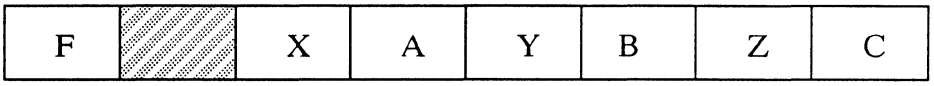
Data Flag Branch Conditions:

Data flag bit 53 — Result field all zeros
Data flag bit 54 — Result field mixed
Data flag bit 55 — Result field all ones

F7

Logical Exclusive OR NOT

Full Word, Format #3
Subfunction: none



A Excl. OR NOT B ---> C

The #F7 instruction performs a bit-by-bit logical exclusive OR NOT function on binary fields A and B. The result is stored in field C. The operation's results, based on bit settings of A and B, are listed below.

Source A B	Result C
0 0	1
0 1	0
1 0	0
1 1	1

The binary fields A, B, and C are strings of bits. The operation proceeds from left to right, terminating when the C field is exhausted. Item counts are bit counts.

If fields A and B are shorter than field C, they are extended automatically with binary zeros. Registers X, Y, and Z contain bit indexes that are added to the A, B, and C addresses, respectively.

Data Flag Branch Conditions:

Data flag bit 53 — Result field all zeros
Data flag bit 54 — Result field mixed
Data flag bit 55 — Result field all ones

F8

Move Bytes Left

Full Word, Format #3
Subfunction: none

A ---> C

The #F8 instruction moves source field A to the result field C. The bytes in the field are considered from left to right, meaning that the most significant byte of the source field is moved to the most significant byte position of the result field.

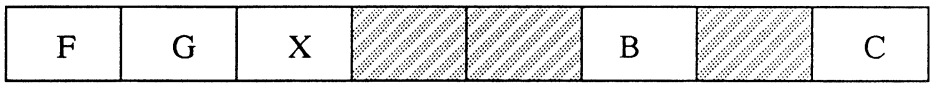
If the source field is shorter than the destination field, the destination field is filled in with the repeated byte found in the B designator. If the source field is longer than the destination field, the operation ends when the destination field is exhausted.

The 48-bit indexes in registers X and Z are left-shifted three bits before being added to the base addresses in registers A and C respectively.

FA

Post Semaphore

Full Word, Format #D
Subfunction: 00ss0000
Qualifiers: ss=[sa0,sa1,sa2,sa3]



The #FA instruction removes a process word from the process queue. Depending on the wait count, bits 0–31 of the process word are returned to bits 0–31 of register C. The wait count is always returned to bits 32–63 of register C. The instruction branches to the CPU branch address in register B if the wait count is equal to or less than –1.

If the wait count is greater than or equal to zero, the following is performed.

1. Calculate the semaphore address by adding the relative bit address (in register X) to the semaphore's base/limit/access selected by the specified *sa0*, *sa1*, *sa2*, or *sa3* qualifier.
2. Read the two-word semaphore from the communication buffer (CB), examine the wait count bits (0–31), and increment the count by one.
3. Store the semaphore back into CB. Load bits 32–63 of register C with the non-updated wait count. Bits 0–31 of register C are zeros.
4. Continue execution at the next sequential instruction.

If the wait count is equal to (–1), the following is performed.

1. Same as for wait count greater than or equal to zero.
2. Same as for wait count greater than or equal to zero.
3. Remove a process word from the queue by reading up the process word located by the Q-head address from CB. Load register C bits 32–63 with the non-updated wait count. Bits 0–31 of register C contains the process word.
4. Store the semaphore back into CB. The new wait count is equal to zero, indicating that the queue is now empty.
5. Branch to the CPU address contained in register B.

If the wait count is less than -1, the following is performed.

1. Same as for wait count greater than or equal to zero.
2. Same as for wait count greater than or equal to zero.
3. Remove a process word from the queue by reading up the process word located by the Q-head address from CB. Load register C bits 32–63 with the non-updated wait count. Bits 0–31 of register C contains the process word.
4. Update the Q-head address with the next process link address from the process word. Store the semaphore back in CB.
5. Branch to the CPU address contained in register B.

FB

Wait on Semaphore

Full Word, Format #D
Subfunction: 00ss00pp
Qualifiers: ss=[sa0,sa1,sa2,sa3],pp=[pa0,pa1,pa2,pa3]

The #FB instruction adds a process word to a process queue, depending on the wait count of the semaphore. The instruction will branch if the wait count is equal to or less than zero. The non-updated wait count is returned to register C.

If the wait count is greater than zero, the following steps are performed.

1. Calculate the semaphore address by adding the relative bit address (in register X) to the semaphore's base/limit/access selected by the specified *sa0*, *sa1*, *sa2*, or *sa3* qualifier.
2. Calculate the new process word address by adding the relative bit address (in register A) to the process word's base/limit/access selected by the specified *pa0*, *pa1*, *pa2*, or *pa3* qualifier.

   Note: This address is sent to the same communication buffer side as in step 1. If the process word address is in the other side, the instruction is undefined. The semaphore may use the process word address from the wrong CB and the instruction may complete with no error indicated.
3. Read the two-word semaphore from CB, examine the wait count bits (0–31), and decrement the wait count by one.
4. Store the semaphore back into CB.
5. Load register C bits 32–63 with the non-updated wait count. Bits 0–31 are zeros.
6. Continue execution at the next sequential instruction.

If the wait count is equal to zero, the following steps are performed.

1. Same as for wait count greater than zero.
2. Same as for wait count greater than zero.
3. Same as for wait count greater than zero.
4. Initialize the process queue by inserting the new process word address (calculated in step 2) into the Q-head and Q-tail address fields of the semaphore. Store back into CB.
5. Load bits 32–63 of register C with the non-updated wait count. Bits 0–31 are zeros.
6. Branch to the CPU address contained in register B.

If the wait count is less than zero, the following steps are performed.

1. Same as for wait count greater than zero.
2. Same as for wait count greater than zero.
3. Same as for wait count greater than zero.
4. Add the new process word to the end of the process queue. This is done by storing the new process word address (calculated in step 2) into the next process link field of the process word located by the Q-tail address of the semaphore.
5. Update the Q-tail address in the semaphore with the new process address in register A before storing it back into CB.
6. Load register C bits 32–63 with the non-updated wait count. Bits 0–31 are zeros.
7. Branch to the CPU address contained in register B.

FC

Bit Branch and Swap

Full Word, Format #D
Subfunction: h00000cc
Qualifiers: h, cc=[ca0, ca1, ca2, ca3]

The #FC instruction performs the following:

1. Calculate the bit branch and swap's bit address by adding the relative bit address (in register X) to the base/limit/access selected by the specified *ca0*, *ca1*, *ca2*, or *ca3* qualifier.
2. Send register A's contents to the communication buffer (CB).
3. Read from CB the word or halfword (if the *h* qualifier is specified) at the address calculated in step 1. Examine the object bit specified by this bit address.

If the object bit is one, execute step 4. If the object bit is zero, execute steps 5, 6, and 7.

4. Branch to the CP address in register B. Clear register C to zeros. Execution of the #FC is now complete.
5. Load into register C the word or half word (if the *h* qualifier is specified) read in step 3.
6. Store register A's contents into the bit branch and swap's bit address calculated in step 1.
7. Continue execution at the next sequential instruction.

If registers A and C are the same, the #FC will execute as above. Upon a second execution, the results may differ because register A has been modified to all zeros.

If the *h* qualifier is not specified, registers A and C are 64 bits. If the *h* qualifier is specified, registers A and C are 32 bits.

FD

Bit Branch and Load/Store

Full Word, Format #D
Subfunction: h00000cc
Qualifiers: h, cc=[ca0, ca1, ca2, ca3]

The #FD instruction performs the following:

1. Calculate the bit branch and load/store's bit address by adding the relative bit address (in register X) to the base/limit/access selected by the specified *ca0*, *ca1*, *ca2*, or *ca3* qualifier.
2. Send register A's contents to the communication buffer (CB).
3. Read from CB the word (halfword if the *h* qualifier is specified) at the address calculated in step 1. Examine the object bit specified by this bit address.

If the object bit is one, execute step 4. If the object bit is zero, execute steps 5, 6, and 7.

4. Branch to the CP address in register B. Clear register C to zeros. Execution of the #FD is now complete
5. Store the word (or half word) from register A into the bit address calculated in step 1.
6. Load into register C the word (or halfword) from the address in step 5 offset by 64 bits (32 if *h* was specified). This is the next sequential word (or half word).
7. Continue execution at the next sequential instruction.

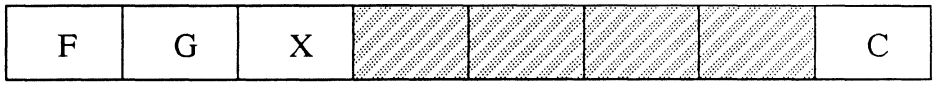
If registers A and C are the same, the #FD executes as described above. Upon a second execution, the results may differ because register A has been modified to all zeros.

If the *h* qualifier is not specified, registers A and C are 64 bits. If the *h* qualifier is specified, registers A and C are 32 bits.

FE

Load Register

Full Word, Format #D
Subfunction: h00000cc
Qualifiers: h,cc=[ca0, ca1, ca2, ca3]



(C) per (X)

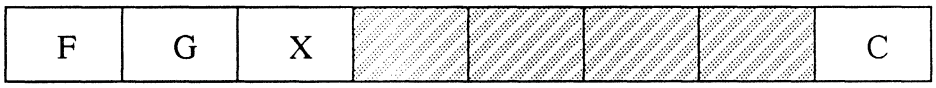
The #FE instruction loads register C with the contents of the CB address calculated by adding the relative bit address (in register X) to the base/limit/access selected by the qualifier *ca0*, *ca1*, *ca2*, or *ca3*.

If the *h* qualifier is specified, register C is 32 bits. If the *h* qualifier is not specified, register C is 64 bits.

FF

Store Register

Full Word, Format #D
Subfunction: h00000cc
Qualifiers: h, cc=[ca0, ca1, ca2, ca3]



(C) per (X)

The #FF instruction stores register C into the CB address calculated by adding the relative bit address (in register X) to the base/limit/access selected by the qualifier *ca0*, *ca1*, *ca2*, or *ca3*.

If the *h* qualifier is specified, register C is 32 bits. If the *h* qualifier is not specified, register C is 64 bits.

Appendix A: Instructions by Function Code

Table A-1. Instructions by Function Code (page 1 of 6).

Function	Format	Mnemonic	G-bits	Operation
00	4	**idle**	--------	Idle
03	4	**nop**	--------	No Operation
04	7	**bkpt**	--------	Breakpoint on Address
05	4	**vsb**	--------	Void Stack and Branch
06	4	**fault**	--------	Fault Test
07	4	**setmod**	--------	Select Serial/Parallel Execution Mode
08	4	**setint**	--------	Transmit External Interrupt
09	4	**exit**	--------	Exit Force
09	4	**exitf**	--------	Exit Force
0A	4	**mtime**	--------	Transmit (R) To Monitor Interval Timer
0C	4	**stoar**	--------	Store Associative Registers
0D	4	**lodar**	--------	Load Associative Registers
0E	4	**rdint**	--------	Read Interrupt Register to (T)
0F	4	**lodkey**	--------	Load Keys from (R), Translate Address (S) to (T)
10	A	**dtob**	--------	Convert BCD to Binary, Fixed Length
11	A	**btod**	--------	Convert Binary to BCD, Fixed Length
12	7	**lodc**	--------	Load Byte from CP memory; (T) Per (S), (R)
13	7	**stoc**	--------	Store Byte to CP memory; (T) Per (S), (R)
14	7	**cpsb**	--------	Bit Compress
15	7	**mrgb**	--------	Bit Merge
16	7	**maskb**	--------	Bit Mask
17	7	**exdom**	--------	Backward Domain Change
18	7	**swcqta**	--------	Shared Memory; CQTA to (T), (S) to CQTA
19	7	**strtio**	--------	Shared Memory; (S) to IQHA, (T) to IQTA
1A	7	**stopio**	--------	Shared Memory; IQHA to (S), IQVF, IQTA to (T)
1B	7	**testio**	--------	Shared Memory; IQVF, Transfer Busy, Fatal Error
1C	7	**maskz**	--------	Form Repeated Bit Mask with Leading Zeros
1D	7	**masko**	--------	Form Repeated Mask with Leading Ones
1E	7	**enteq**	--------	Count Leading Equals
1F	7	**ento**	--------	Count Ones in Field R, Count to (T)
20	8	**bheq**	--------	Branch if (R) Equal (S) (32-Bit)
21	8	**bhne**	--------	Branch if (R) Not Equal (S) (32-Bit)
22	8	**bhge**	--------	Branch if (R) Greater or Equal (S) (32-Bit)
23	8	**bhlt**	--------	Branch if (R) Less Than (S) (32-Bit)
24	8	**beq**	--------	Branch if (R) Equal (S) (64-Bit)
25	8	**bne**	--------	Branch if (R) Not Equal (S) (64-Bit)
26	8	**bge**	--------	Branch if (R) Greater or Equal (S) (64-Bit)
27	8	**blt**	--------	Branch if (R) Less Than (S) (64-Bit)
28	7	**scnleq**	--------	Scan for Equal Byte

Table A-2. Instructions by Function Code (page 2 of 6).

Function	Format	Mnemonic	G-bits	Operation
29	7	**tfc**	--------	Transmit Instrumentation Counter to (T)
2A	6	**elen**	--------	Enter Length of (R) with I (16 Bits)
2B	4	**addlen**	--------	Add to Length Field
2C	4	**rxor**	--------	Logical Exclusive OR (R), (S) to (T)
2D	4	**rand**	--------	Logical AND (R), (S) to (T)
2E	4	**rior**	--------	Logical Inclusive OR (R), (S) to (T)
2F	9	**barb**	--------	Register Bit Branch and Alter
30	7	**shifti**	--------	Shift Operand; (R) per S to (T)
31	7	**ibnz**	--------	Increase (R) and Branch if (R) NE 0
32	9	**bab**	--------	Bit Branch and Alter
33	B	**badf**	--------	Data Flag Register Bit Branch and Alter
34	4	**shift**	--------	Shift (R) per S to (T)
35	7	**dbnz**	--------	Decrease (R) and Branch if (R) NE 0
36	7	**bsave**	--------	Branch or Forward Domain Change
37	A	**rjtime**	--------	Transmit Job Interval Timer to (T)
38	A	**ltol**	--------	Transmit (R) Bits 0-15 to (T) Bits 0-15
39	A	**clock**	--------	Transmit Real Time Clock to (T)
3A	A	**wjtime**	--------	Transmit (R) to Job Interval Timer
3B	A	**lsdfr**	--------	Data Flag Register Load/Store
3C	4	**mpyxh**	--------	Half-Word Index Multiply (R)*(S) to (T)
3D	4	**mpyx**	--------	Index Multiply (R)*(S) to (T)
3E	6	**es**	--------	Enter (R) with I (16 Bits)
3F	6	**is**	--------	Increase (R) By I (16 Bits)
40	4	**adduh**	--------	Add; Upper result (R) + (S) to (T) (32 Bits)
41	4	**addlh**	--------	Add; Lower result (R) + (S) to (T) (32 Bits)
42	4	**addnh**	--------	Add; Normalized result (R) + (S) to (T) (32 Bits)
44	4	**subuh**	--------	Subtract; Upper result (R) - (S) to (T) (32 Bits)
45	4	**sublh**	--------	Subtract; Lower result (R) - (S) to (T) (32 Bits)
46	4	**subnh**	--------	Subtract; Normalized result (R) - (S) to (T) (32 Bits)
48	4	**mpyuh**	--------	Multiply; Upper result (R) * (S) to (T) (32 Bits)
49	4	**mpylh**	--------	Multiply; Lower result (R) * (S) to (T) (32 Bits)
4B	4	**mpysh**	--------	Multiply; Significant result (R) * (S) to (T) (32 Bits)
4C	4	**divuh**	--------	Divide; Upper result (R) / (S) to (T) (32 Bits)
4D	6	**esh**	--------	Half-Word Enter (R) with I (16 Bits)
4E	6	**ish**	--------	Half-Word Increase (R) By I (16 Bits)
4F	4	**divsh**	--------	Divide; Significant result (R) / (S) to (T) (32 Bits)
50	A	**truh**	--------	Truncate; (R) to (T) (32 Bits)
51	A	**flrh**	--------	Floor; (R) to (T) (32 Bits)
52	A	**clgh**	--------	Ceiling; (R) to (T) (32 Bits)
53	A	**sqrth**	--------	Significant Square Root; (R) to (T) (32 Bits)
54	4	**adjsh**	--------	Adjust Significance; (R) per (S) to (T) (32 Bits)
55	4	**adjeh**	--------	Adjust Exponent; (R) per (S) to (T) (32 Bits)
56	7	**linkv**	--------	Select Link
57	7	**rddom**	--------	Read Domain Registers; Special Register per R to (T)
58	A	**rtorh**	--------	Transmit Operand; (R) to (T) (32 Bits)
59	A	**absh**	--------	Transmit Absolute; (R) to (T) (32 Bits)
5A	A	**exph**	--------	Transmit Exponent; (R) to (T) (32 Bits)
5B	4	**packh**	--------	Pack; (R), (S) to (T) (32 Bits)
5C	A	**exth**	--------	Extend; 32-Bit (R) to 64-Bit (T)

Table A-3. Instructions by Function Code (page 3 of 6).

Function	Format	Mnemonic	G-bits	Operation
5D	A	**extxh**	--------	Index Extend; 32-Bit (R) to 64-Bit (T)
5E	7	**lodh**	--------	Load; (T) per (S), (R) (Halfword)
5F	7	**stoh**	--------	Store; (T) per (S), (R) (Halfword)
60	4	**addu**	--------	Add; Upper result (R) + (S) to (T) (64 Bits)
61	4	**addl**	--------	Add; Lower result (R) + (S) to (T) (64 Bits)
62	4	**addn**	--------	Add; Normalized result (R) + (S) to (T) (64 Bits)
63	4	**addx**	--------	Add Address; (R) + (S) to (T)
64	4	**subu**	--------	Subtract; Upper result (R) - (S) to (T) (64 Bits)
65	4	**subl**	--------	Subtract; Lower result (R) - (S) to (T) (64 Bits)
66	4	**subn**	--------	Subtract; Normalized result (R) - (S) to (T) (64 Bits)
67	4	**subx**	--------	Subtract Address; (R) - (S) to (T)
68	4	**mpyu**	--------	Multiply; Upper result (R) * (S) to (T) (64 Bits)
69	4	**mpyl**	--------	Multiply; Lower result (R) * (S) to (T) (64 Bits)
6B	4	**mpys**	--------	Multiply; Significant result (R) * (S) to (T) (64 Bits)
6C	4	**divu**	--------	Divide; Upper result (R) / (S) to (T) (64 Bits)
6D	4	**insb**	--------	Insert Bits; (R) to (T) per (S)
6E	4	**extb**	--------	Extract Bits; (R) to (T) per (S)
6F	4	**divs**	--------	Divide; Significant result (R) / (S) to (T) (64 Bits)
70	A	**tru**	--------	Truncate; (R) to (T) (64 Bits)
71	A	**flr**	--------	Floor; (R) to (T) (64 Bits)
72	A	**elg**	--------	Ceiling; (R) to (T) (64 Bits)
73	A	**sqrt**	--------	Significant Square Root; (R) to (T) (64 Bits)
74	4	**adjs**	--------	Adjust Significance; (R) per (S) to (T) (64 Bits)
75	4	**adje**	--------	Adjust Exponent; (R) per (S) to (T) (64 Bits)
76	A	**con**	--------	Contract; 64-Bit (R) to 32-Bit (T)
77	A	**rcon**	--------	Rounded Contract; 64-Bit (R) to 32-Bit (T)
78	A	**rtor**	--------	Transmit; (R) to (T) (64 Bits)
79	A	**abs**	--------	Absolute; (R) to (T) (64 Bits)
7A	A	**exp**	--------	Exponent; (R) to (T) (64 Bits)
7B	4	**pack**	--------	Pack; (R), (S) to (T) (64 Bits)
7C	A	**ltor**	--------	Length; (R) to (T) (64 Bits)
7D	4	**rgap**	--------	Swap; S ---> T and R ---> S
7E	7	**lod**	--------	Load; (T) per (S), (R) (Word)
7F	7	**sto**	--------	Store; (T) per (S), (R) (Word)
80	1	**adduv**	**hzoabsss**	Add; Upper result A + B ---> C
81	1	**addlv**	**hzoabsss**	Add; Lower result A + B ---> C
82	1	**addnv**	**hzoabsss**	Add; Normalized result A + B ---> C
83	1	**addxv**	**0zoab000**	Add Address; A + B ---> C
84	1	**subuv**	**hzoabsss**	Subtract; Upper result A - B ---> C
85	1	**sublv**	**hzoabsss**	Subtract; Lower result A - B ---> C
86	1	**subnv**	**hzoabsss**	Subtract; Normalized result A - B ---> C
87	1	**subxv**	**0zoab000**	Subtract Address; A - B ---> C
88	1	**mpyuv**	**hzoabsss**	Multiply; Upper result A * B ---> C
89	1	**mpylv**	**hzoabsss**	Multiply; Lower result A * B ---> C
8A	1	**shiftv**	**0zoab000**	Shift; A per B ---> C
8B	1	**mpysv**	**hzoabsss**	Multiply; Significant result A * B ---> C
8C	1	**divuv**	**hzoabsss**	Divide; Upper result A / B ---> C
8F	1	**divsv**	**hzoabsss**	Divide; Significant result A / B ---> C
90	1	**truv**	**hzoa0000**	Truncate; A ---> C

Table A-4. Instructions by Function Code (page 4 of 6).

Function	Format	Mnemonic	G-bits	Operation
91	1	**flrv**	**hzoa0000**	Floor: A ---> C
92	1	**clgv**	**hzoa0000**	Ceiling: A ---> C
93	1	**sqrtv**	**hzoa0ss0**	Significant Square Root; A ---> C
94	1	**adjsv**	**hzoab000**	Adjust Significance; A per B ---> C
95	1	**adjev**	**hzoab000**	Adjust Exponent; A per B ---> C
96	1	**conv**	**0zoa0000**	Contract; 64-Bit A ---> 32-Bit C
97	1	**rconv**	**0zoa0000**	Rounded Contract; 64-Bit A ---> 32-Bit C
98	1	**vtov**	**hzoa0000**	Transmit Element; A ---> C
99	1	**absv**	**hzoa0000**	Move Absolute; A ---> C
9A	1	**expv**	**hzoa0000**	Move Exponent; A ---> C
9B	1	**packv**	**hzoab000**	Pack; A, B ---> C
9C	1	**extv**	**0zoa0000**	Extend; 32-Bit A ---> 64-Bit C
9D	1	**andnv**	**hzoabnnn**	Logical AND NOT; A, B, ---> C
9D	1	**andv**	**hzoabnnn**	Logical AND; A, B, ---> C
9D	1	**iorv**	**hzoabnnn**	Logical Inclusive OR; A, B, ---> C
9D	1	**nandv**	**hzoabnnn**	Logical NOT AND; A, B, ---> C
9D	1	**norv**	**hzoabnnn**	Logical NOT OR; A, B, ---> C
9D	1	**ornv**	**hzoabnnn**	Logical OR NOT; A, B, ---> C
9D	1	**xornv**	**hzoabnnn**	Logical Exclusive OR NOT; A, B, ---> C
9D	1	**xorv**	**hzoabnnn**	Logical Exclusive OR; A, B, ---> C
A0	2	**addus**	**hllabsss**	Add; Upper result A + B ---> C
A1	2	**addls**	**hllabsss**	Add; Lower result A + B ---> C
A2	2	**addns**	**hllabsss**	Add N; A + B ---> C
A4	2	**subus**	**hllabsss**	Subtract; Upper result A - B ---> C
A5	2	**subls**	**hllabsss**	Subtract; Lower result A - B ---> C
A6	2	**subns**	**hllabsss**	Subtract N; A - B ---> C
A8	2	**mpyus**	**hllabsss**	Multiply; Upper result A * B ---> C
A9	2	**mpyls**	**hllabsss**	Multiply; Lower result A * B ---> C
AB	2	**mpyss**	**hllabsss**	Multiply; Significant result A * B ---> C
AC	2	**divus**	**hllabsss**	Divide; Upper result A / B ---> C
AF	2	**divss**	**hllabsss**	Divide; Significant result A / B ---> C
B0	C	**cfpeq**	**h1c00000**	Compare F.P., Set Condition if (A) EQ (X)
B0	C	**cfpeq**	**h1000bb0**	Compare F.P., Branch if (A) EQ (X)
B0	C	**ibxeq**	**h0cfu000**	Compare Integers, Set Condition if (A)+(X) EQ (Z)
B0	C	**ibxeq**	**h00fubb0**	Compare Integers, Branch if (A) EQ (Z)
B1	C	**cfpne**	**h1c00000**	Compare F.P., Set Condition if (A) NE (X)
B1	C	**cfpne**	**h1000bb0**	Compare F.P., Branch if (A) NE (X)
B1	C	**ibxne**	**h0cfu000**	Compare Integers, Set Condition if (A)+(X) NE (Z)
B1	C	**ibxne**	**h00fubb0**	Compare Integers, Branch if (A)+(X) NE (Z)
B2	C	**cfpge**	**h1c00000**	Compare F.P., Set Condition if (A) GE (X)
B2	C	**cfpge**	**h1000bb0**	Compare F.P., Branch if (A) GE (X)
B2	C	**ibxge**	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) GE (Z)
B2	C	**ibxge**	**h000ubb0**	Compare Integers, Branch if (A)+(X) GE (Z)
B3	C	**cfplt**	**h1c00000**	Compare F.P., Set Condition if (A) LT (X)
B3	C	**cfplt**	**h1000bb0**	Compare F.P., Branch if (A) LT (X)
B3	C	**ibxlt**	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) LT (Z)
B3	C	**ibxlt**	**h000ubb0**	Compare Integers, Branch if (A)+(X) LT (Z)
B4	C	**cfple**	**h1c00000**	Compare F.P., Set Condition if (A) LE (X)
B4	C	**cfple**	**h1000bb0**	Compare F.P., Branch if (A) LE (X)

Table A-5. Instructions by Function Code (page 5 of 6).

Function	Format	Mnemonic	G-bits	Operation
B4	C	**ibxle**	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) LE (Z)
B4	C	**ibxle**	**h000ubb0**	Compare Integers, Branch if (A)+(X) LE (Z)
B5	C	**cfpgt**	**h1c00000**	Compare F.P., Set Condition if (A) GT (X)
B5	C	**cfpgt**	**h1000bb0**	Compare F.P., Branch if (A) GT (X)
B5	C	**ibxgt**	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) GT (Z)
B5	C	**ibxgt**	**h000ubb0**	Compare Integers, Branch if (A)+(X) GT (Z)
B6	5	**bim**	--------	Branch to Immediate Address; (R) + I (48 Bits)
B7	1	**vtovx**	**h000bfgr**	Scatter ---> Indexed C
B8	1	**vrevv**	**hzo00000**	Transmit Reverse; A ---> C
BA	1	**vxtov**	**h0000fgr**	Gather ---> C
BB	2	**maskv**	**h00ab000**	Mask; A, B ---> C per Z
BC	2	**cpsv**	**hz000000**	Compress; A ---> C per Z
BD	2	**mrgv**	**h00ab00s**	Merge; A, B ---> C per Z
BE	5	**ex**	--------	Enter (R) with I (48 Bits)
BF	5	**ix**	--------	Increase (R) By I (48 Bits)
C0	1	**seleq**	**hz0ab000**	Select Equal; A EQ B, Item Count to (C)
C1	1	**selne**	**hz0ab000**	Select Not Equal; A NE B, Item Count to (C)
C2	1	**selge**	**hz0ab000**	Select Greater or Equal; A GE B, Item Count to (C)
C3	1	**sellt**	**hz0ab000**	Select Less; A LT B, Item Count to (C)
C4	1	**cmpeq**	**h00ab000**	Compare Equal; A EQ B Order Vector ---> Z
C5	1	**cmpne**	**h00ab000**	Compare Not Equal; A NE B Order Vector ---> Z
C6	1	**cmpge**	**h00ab000**	Compare GE; A GE B Order Vector ---> Z
C7	1	**cmplt**	**h00ab000**	Compare Less; A LT B Order Vector ---> Z
C8	1	**srcheq**	**hzl00000**	Search for Equality; Index List ---> C
C9	1	**srchne**	**hzl00000**	Search for Inequality; Index List ---> C
CA	1	**srchge**	**hzl00000**	Search for Greater or Equal; Index List ---> C
CB	1	**srchlt**	**hzl00000**	Search for Less; Index List ---> C
CC	D	**mcmpw**	**0000000n**	Masked Binary Compare; A EQ/NE (B) per (C)
CD	5	**exh**	--------	Half-Word Enter (R) By I (24 Bits)
CE	5	**ixh**	--------	Half-Word Increase (R) By I (24 Bits)
CF	1	**acps**	**h000bsss**	Arithmetic Compress; A ---> C per B
CF	1	**aricps**	**h000bsss**	Arithmetic Compress; A ---> C per B
CF	1	**arithcps**	**h000bsss**	Arithmetic Compress; A ---> C per B
D0	1	**avg**	**hzoab000**	Average; (A(N) + B(N))/2 ---> C(N)
D1	1	**adjmean**	**hzo00000**	Adjacent Mean; (A(N+1) - A(N))/2 ---> C(N)
D4	1	**avgd**	**hzoab000**	Average Difference; (A(N) - B(N))/2 ---> C(N)
D5	1	**delta**	**hzo00000**	Delta; (A(N+1)-A(N)) ---> C(N)
D8	1	**max**	**hz000s00**	Maximum of Vector A to (C), Item Count to (B)
D9	1	**min**	**hz000s00**	Minimum of Vector A to (C), Item Count to (B)
DA	1	**sum**	**hz000000**	Sum; (A0+A1+A2+ ...+n) To (C) and (C+1)
DB	1	**product**	**hz000000**	Product; (A0*A1*A2*A3 ...*An) To (C)
DC	1	**dotv**	**hz0ab000**	Dot Product to (C) and (C+1)
DF	1	**interval**	**hzo00000**	Interval; (A) per (B) ---> C
DF	1	**intrval**	**hzo00000**	Interval; (A) per (B) ---> C
DF	1	**intval**	**hzo00000**	Interval; (A) per (B) ---> C
F0	3	**xor**	--------	Logical Exclusive OR; A, B ---> C
F1	3	**and**	--------	Logical AND; A, B ---> C
F2	3	**ior**	--------	Logical Inclusive OR; A,B ---> C
F3	3	**nand**	--------	Logical NOT AND; A,B ---> C

Table A-6. Instructions by Function Code (page 6 of 6).

Function	Format	Mnemonic	G-bits	Operation
F4	3	**nor**	--------	Logical NOT OR; A,B ---> C
F5	3	**orn**	--------	Logical Exclusive OR NOT; A,B ---> C
F6	3	**andn**	--------	Logical AND NOT; A,B ---> C
F7	3	**xorn**	--------	Logical Exclusive OR NOT; A,B ---> C
F8	3	**movl**	--------	Move Bytes Left; A ---> C
FA	D	**post**	**00ss0000**	Post Semaphore
FB	D	**wait**	**00ss00pp**	Wait on Semaphore
FC	D	**bbswap**	**h00000cc**	Bit Branch and Swap
FD	D	**bbldst**	**h00000cc**	Bit Branch and Load/Store
FE	D	**cblod**	**h00000cc**	Load Register; (C) per (X)
FF	D	**cbsto**	**h00000cc**	Store Register; (C) per (X)

Appendix B: Instructions by Mnemonic

Table B-1. Instructions by Mnemonic (page 1 of 6).

Mnemonic	Format	Function	G-bits	Operation
abs	A	79	--------	Absolute; (R) to (T)
absh	A	59	--------	Transmit Absolute; (R) to (T)
absv	1	99	**hzoa0000**	Move Absolute; A ---> C
acps	1	CF	**h000bsss**	Arithmetic Compress; A ---> C per B
addl	4	61	--------	Add; Lower result (R) + (S) to (T) (64 Bits)
addlen	4	2B	--------	Add to Length Field
addlh	4	41	--------	Add; Lower result (R) + (S) to (T) (32 Bits)
addls	2	A1	**hllabsss**	Add; Lower result A + B ---> C
addlv	1	81	**hzoabsss**	Add; Lower result A + B ---> C
addn	4	62	--------	Add; Normalized result (R) + (S) to (T) (64 Bits)
addnh	4	42	--------	Add; Normalized result (R) + (S) to (T) (32 Bits)
addns	2	A2	**hllabsss**	Add; Normalized result A + B ---> C
addnv	1	82	**hzoabsss**	Add; Normalized result A + B ---> C
addu	4	60	--------	Add; Upper result (R) + (S) to (T) (64 Bits)
adduh	4	40	--------	Add; Upper result (R) + (S) to (T) (32 Bits)
addus	2	A0	**hllabsss**	Add; Upper result A + B ---> C
adduv	1	80	**hzoabsss**	Add; Upper result A + B ---> C
addx	4	63	--------	Add Address; (R) + (S) to (T)
addxv	1	83	**0zoab000**	Add Address; A + B ---> C
adje	4	75	--------	Adjust Exponent; (R) per (S) to (T)
adjeh	4	55	--------	Adjust Exponent; (R) per (S) to (T)
adjev	1	95	**hzoab000**	Adjust Exponent; A per B ---> C
adjmean	1	D1	**hzo00000**	Adjacent Mean; (A(N+1) - A(N))/2 ---> C(N)
adjs	4	74	--------	Adjust Significance; (R) per (S) to (T)
adjsh	4	54	--------	Adjust Significance; (R) per (S) to (T)
adjsv	1	94	**hzoab000**	Adjust Significance; A per B ---> C
and	3	F1	--------	Logical AND; A, B ---> C
andn	3	F6	--------	Logical AND NOT; A,B ---> C
andnv	1	9D	**hzoabnnn**	Logical AND NOT; A, B, ---> C
andv	1	9D	**hzoabnnn**	Logical AND; A, B, ---> C
aricps	1	CF	**h000bsss**	Arithmetic Compress; A ---> C per B
arithcps	1	CF	**h000bsss**	Arithmetic Compress; A ---> C per B
avg	1	D0	**hzoab000**	Average; (A(N) + B(N))/2 ---> C(N)
avgd	1	D4	**hzoab000**	Average Difference; (A(N) - B(N))/2 ---> C(N)
bab	9	32	--------	Bit Branch and Alter
badf	B	33	--------	Data Flag Register Bit Branch and Alter
barb	9	2F	--------	Register Bit Branch and Alter
bbldst	D	FD	**h00000cc**	Bit Branch and Load/Store
bbswap	D	FC	**h00000cc**	Bit Branch and Swap
beq	8	24	--------	Branch if (R) Equal (S) (64-Bit)

Table B-2. Instructions by Mnemonic (page 2 of 6).

Mnemonic	Format	Function	G-bits	Operation
bge	8	26	--------	Branch if (R) Greater or equal (S) (64-Bit)
bheq	8	20	--------	Branch if (R) Equal (S) (32-Bit)
bhge	8	22	--------	Branch if (R) Greater or Equal (S) (32-Bit)
bhlt	8	23	--------	Branch if (R) Less Than (S) (32-Bit)
bhne	8	21	--------	Branch if (R) Not Equal (S) (32-Bit)
bim	5	B6	--------	Branch to Immediate Address; (R) + I (48 Bits)
bkpt	7	04	---------	Breakpoint on Address
blt	8	27		Branch if (R) Less Than (S) (64-Bit)
bne	8	25	--------	Branch if (R) Not Equal (S) (64-Bit)
bsave	7	36	--------	Branch or Forward Domain Change
btod	A	11	--------	Convert Binary to BCD, Fixed Length
cblod	D	FE	**h00000cc**	Load Register; (C) per (X)
cbsto	D	FF	**h00000cc**	Store Register; (C) per (X)
cfpeq	C	B0	**h1c00000**	Compare F.P., Set Condition if (A) EQ (X)
cfpeq	C	B0	**h1000bb0**	Compare F.P., Branch if (A) EQ (X)
cfpge	C	B2	**h1c00000**	Compare F.P., Set Condition if (A) GE (X)
cfpge	C	B2	**h1000bb0**	Compare F.P., Branch if (A) GE (X)
cfpgt	C	B5	**h1c00000**	Compare F.P., Set Condition if (A) GT (X)
cfpgt	C	B5	**h1000bb0**	Compare F.P., Branch if (A) GT (X)
cfple	C	B4	**h1c00000**	Compare F.P., Set Condition if (A) LE (X)
cfple	C	B4	**h1000bb0**	Compare F.P., Branch if (A) LE (X)
cfplt	C	B3	**h1c00000**	Compare F.P., Set Condition if (A) LT (X)
cfplt	C	B3	**h1000bb0**	Compare F.P., Branch if (A) LT (X)
cfpne	C	B1	**h1c00000**	Compare F.P., Set Condition if (A) NE (X)
cfpne	C	B1	**h1000bb0**	Compare F.P., Branch if (A) NE (X)
clg	A	72	--------	Ceiling; (R) to (T)
clgh	A	52	--------	Ceiling; (R) to (T)
clgv	1	92	**hzoa0000**	Ceiling: A ---> C
clock	A	39	--------	Transmit Real Time Clock to (T)
cmpeq	1	C4	**h00ab000**	Compare Equal; A = B Order Vector ---> Z
cmpge	1	C6	**h00ab000**	Compare GE; A GE B Order Vector ---> Z
cmplt	1	C7	**h00ab000**	Compare Less; A LT B Order Vector ---> Z
cmpne	1	C5	**h00ab000**	Compare Not Equal; A NE B Order Vector ---> Z
con	A	76	--------	Contract; 64-Bit (R) to 32-Bit (T)
conv	1	96	**0zoa0000**	Contract; 64-Bit A ---> 32-Bit C
cpsb	7	14	--------	Bit Compress
cpsv	2	BC	**hz000000**	Compress; A ---> C Per Z
dbnz	7	35	--------	Decrease (R) and Branch if (R) NE 0
delta	1	D5	**hzo00000**	Delta; (A(N+1)-A(N)) ---> C(N)
divs	4	6F	--------	Divide; Significant result (R) / (S) to (T) (64 Bits)
divsh	4	4F	--------	Divide; Significant result (R) / (S) to (T) (32 Bits)
divss	2	AF	**hllabsss**	Divide; Significant result A / B ---> C
divsv	1	8F	**hzoabsss**	Divide; Significant result A / B ---> C
divu	4	6C	--------	Divide; Upper result (R) / (S) to (T) (64 Bits)
divuh	4	4C	--------	Divide; Upper result (R) / (S) to (T) (32 Bits)
divus	2	AC	**hllabsss**	Divide; Upper result A / B ---> C
divuv	1	8C	**hzoabsss**	Divide; Upper result A / B ---> C

Table B-3. Instructions by Mnemonic (page 3 of 6).

Mnemonic	Format	Function	G-bits	Operation
dotv	1	DC	**hz0ab000**	Dot Product to (C) and (C+1)
dtob	A	10	--------	Convert BCD to Binary, Fixed Length
elen	6	2A	--------	Enter Length of (R) With I (16 Bits)
enteq	7	1E	--------	Count Leading Equals
ento	7	1F	--------	Count Ones in Field R, Count to (T)
es	6	3E	--------	Enter (R) With I (16 Bits)
esh	6	4D	--------	Half-Word Enter (R) With I (16 Bits)
ex	5	BE	--------	Enter (R) With I (48 Bits)
exdom	7	17	--------	Backward Domain Change
exh	5	CD	--------	Half-Word Enter (R) By I (24 Bits)
exit	4	09	--------	Exit Force
exitf	4	09	--------	Exit Force
exp	A	7A	--------	Exponent; (R) to (T)
exph	A	5A	--------	Transmit Exponent; (R) to (T)
expv	1	9A	**hzoa0000**	Move Exponent; A ---> C
extb	4	6E	--------	Extract Bits; (R) to (T) per (S)
exth	A	5C		Extend; 32-Bit (R) to 64-Bit (T)
extv	1	9C	**0zoa0000**	Extend; 32-Bit A ---> 64-Bit C
extxh	A	5D	--------	Index Extend; 32-Bit (R) to 64-Bit (T)
fault	4	06	--------	Fault Test
flr	A	71	--------	Floor; (R) to (T)
flrh	A	51	--------	Floor; (R) to (T)
flrv	1	91	**hzoa0000**	Floor: A ---> C
ibnz	7	31	--------	Increase (R) and Branch if (R) NE 0
ibxeq	C	B0	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) EQ (Z)
ibxeq	C	B0	**h000ubb0**	Compare Integers, Branch if (A)+(X) EQ (Z)
ibxge	C	B2	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) GE (Z)
ibxge	C	B2	**h000ubb0**	Compare Integers, Branch if (A)+(X) GE (Z)
ibxgt	C	B5	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) GT (Z)
ibxgt	C	B5	**h000ubb0**	Compare Integers, Branch if (A)+(X) GT (Z)
ibxle	C	B4	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) LE (Z)
ibxle	C	B4	**h000ubb0**	Compare Integers, Branch if (A)+(X) LE (Z)
ibxlt	C	B3	**h0c0u000**	Compare Integers, Set Condition if (A)+(X) LT (Z)
ibxlt	C	B3	**h000ubb0**	Compare Integers, Branch if (A)+(X) LT (Z)
ibxne	C	B1	**h0cfu000**	Compare Integers, Set Condition if (A)+(X) NE (Z)
ibxne	C	B1	**h00fubb0**	Compare Integers, Branch if (A)+(X) NE (Z)
idle	4	00	--------	Idle
insb	4	6D	--------	Insert Bits; (R) to (T) per (S)
interval	1	DF	**hzo00000**	Interval; (A) per (B) ---> C
intrval	1	DF	**hzo00000**	Interval; (A) per (B) ---> C
intval	1	DF	**hzo00000**	Interval; (A) per (B) ---> C
ior	3	F2	--------	Logical Inclusive OR; A,B ---> C
iorv	1	9D	**hzoabnnn**	Logical Inclusive OR; A, B, ---> C
is	6	3F	--------	Increase (R) By I (16 Bits)
ish	6	4E	--------	Half-Word Increase (R) By I (16 Bits)
ix	5	BF	--------	Increase (R) By I (48 Bits)
ixh	5	CE	--------	Half-Word Increase (R) By I (24 Bits)
linkv	7	56	--------	Select Link
lod	7	7E	--------	Load; (T) per (S), (R)

Table B-4. Instructions by Mnemonic (page 4 of 6).

Mnemonic	Format	Function	G-bits	Operation
lodar	4	0D	--------	Load Associative Registers
lodc	7	12	--------	Load Byte; (T) Per (S), (R)
lodh	7	5E	--------	Load; (T) Per (S), (R)
lodkey	4	0F	--------	Load Keys from (R), Translate Address (S) to (T)
lsdfr	A	3B	--------	Data Flag Register Load/Store
ltol	A	38	--------	Transmit (R) Bits 0-15 to (T) Bits 0-15
ltor	A	7C	--------	Length; (R) to (T)
maskb	7	16	--------	Bit Mask
masko	7	1D	--------	Form Repeated Mask With Leading Ones
maskv	2	BB	**h00ab000**	Mask; A, B ---> C Per Z
maskz	7	1C	--------	Form Repeated Bit Mask With Leading Zeros
max	1	D8	**hz000s00**	Maximum of Vector A to (C), Item Count to (B)
mcmpw	D	CC	**0000000n**	Masked Binary Compare; A EQ/NE (B) Per (C)
min	1	D9	**hz000s00**	Minimum of Vector A to (C), Item Count to (B)
movl	3	F8	--------	Move Bytes Left; A ---> C
mpyl	4	69	--------	Multiply; Lower result (R) * (S) to (T) (64 Bits)
mpylh	4	49	--------	Multiply; Lower result (R) * (S) to (T) (32 Bits)
mpyls	2	A9	**hllabsss**	Multiply; Lower result A * B ---> C
mpylv	1	89	**hzoabsss**	Multiply; Lower result A * B ---> C
mpys	4	6B	--------	Multiply; Significant result (R) * (S) to (T) (64 Bits)
mpysh	4	4B	--------	Multiply; Significant result (R) * (S) to (T) (32 Bits)
mpyss	2	AB	**hllabsss**	Multiply; Significant result A * B ---> C
mpysv	1	8B	**hzoabsss**	Multiply; Significant result A * B ---> C
mpyu	4	68	--------	Multiply; Upper result (R) * (S) to (T) (64 Bits)
mpyuh	4	48	--------	Multiply; Upper result (R) * (S) to (T) (32 Bits)
mpyus	2	A8	**hllabsss**	Multiply; Upper result A * B ---> C
mpyuv	1	88	**hzoabsss**	Multiply; Upper result A * B ---> C
mpyx	4	3D	--------	Index Multiply (R)*(S) to (T)
mpyxh	4	3C	--------	Half-Word Index Multiply (R)*(S) to (T)
mrgb	7	15	--------	Bit Merge
mrgv	2	BD	**h00ab00s**	Merge; A, B ---> C Per Z
mtime	4	0A	--------	Transmit (R) To Monitor Interval Timer
nand	3	F3	--------	Logical NOT AND; A,B ---> C
nandv	1	9D	**hzoabnnn**	Logical NOT AND; A, B, ---> C
nop	4	03	--------	No Operation
nor	3	F4	--------	Logical NOT OR; A,B ---> C
norv	1	9D	**hzoabnnn**	Logical NOT OR; A, B, ---> C
orn	3	F5	--------	Logical OR NOT; A,B ---> C
ornv	1	9D	**hzoabnnn**	Logical OR NOT; A, B, ---> C
pack	4	7B	--------	Pack; (R), (S) to (T)
packh	4	5B	--------	Pack; (R), (S) to (T)
packv	1	9B	**hzoab000**	Pack; A, B ---> C
post	D	FA	**00ss0000**	Post Semaphore
product	1	DB	**hz000000**	Product; (A0*A1*A2*A3 ...*An) To (C)
rand	4	2D	--------	Logical AND (R), (S) to (T)
rcon	A	77	--------	Rounded Contract; 64-Bit (R) to 32-Bit (T)
rconv	1	97	**0zoa0000**	Rounded Contract; 64-Bit A ---> 32-Bit C
rddom	7	57	--------	Read Domain Registers; Special Register Per R to (T)
rdint	4	0E	--------	Read Interrupt Register

Table B-5. Instructions by Mnemonic (page 5 of 6).

Mnemonic	Format	Function	G-bits	Operation
rgap	4	7D	--------	Swap; S ---> T and R ---> S
rior	4	2E	--------	Logical Inclusive OR (R), (S) to (T)
rjtime	A	37	--------	Transmit Job Interval Timer to (T)
rtor	A	78	--------	Transmit; (R) to (T)
rtorh	A	58	--------	Transmit Operand; (R) to (T)
rxor	4	2C	--------	Logical Exclusive OR (R), (S) to (T)
scnleq	7	28		Scan for Equal Byte
seleq	1	C0	**hz0ab000**	Select Equal; A EQ B, Item Count to (C)
selge	1	C2	**hz0ab000**	Select Greater or Equal; A GE B, Item Count to (C)
sellt	1	C3	**hz0ab000**	Select Less; A LT B, Item Count to (C)
selne	1	C1	**hz0ab000**	Select Not Equal; A NE B, Item Count to (C)
setint	4	08	--------	Transmit External Interrupt
setmod	4	07	--------	Serial/Parallel Execution Mode Select
shift	4	34	--------	Shift (R) Per (S) to (T)
shifti	7	30	--------	Shift Operands (R) Per S to (T)
shiftv	1	8A	**0zoab000**	Shift; A Per B ---> C
sqrt	A	73	--------	Significant Square Root; (R) to (T) (64 Bits)
sqrth	A	53	--------	Significant Square Root; (R) to (T)
sqrtv	1	93	**hzoa0ss0**	Significant Square Root; A ---> C
srcheq	1	C8	**hzl00000**	Search for Equality; Index List ---> C
srchge	1	CA	**hzl00000**	Search for Greater or Equal; Index List ---> C
srchlt	1	CB	**hzl00000**	Search for Less; Index List ---> C
srchne	1	C9	**hzl00000**	Search for Not Equal; Index List ---> C
sto	7	7F	--------	Store; (T) Per (S), (R)
stoar	4	0C	--------	Store Associative Registers
stoc	7	13	--------	Store Byte; (T) Per (S), (R)
stoh	7	5F	--------	Store; (T) Per (S), (R)
stopio	7	1A	--------	Shared Memory; IQHA to (S), IQVF, IQTA to (T)
strtio	7	19	--------	Shared Memory; (S) to IQHA, (T) to IQTA
subl	4	65	--------	Subtract; Lower result (R) - (S) to (T) (64 Bits)
sublh	4	45	--------	Subtract; Lower result (R) - (S) to (T) (32 Bits)
subls	2	A5	**hllabsss**	Subtract; Lower result A - B ---> C
sublv	1	85	**hzoabsss**	Subtract; Lower result A - B ---> C
subn	4	66	--------	Subtract; Normalized result (R) - (S) to (T) (64 Bits)
subnh	4	46	--------	Subtract; Normalized result (R) - (S) to (T) (32 Bits)
subns	2	A6	**hllabsss**	Subtract; Normalized result A - B ---> C
subnv	1	86	**hzoabsss**	Subtract; Normalized result A - B ---> C
subu	4	64	--------	Subtract; Upper result (R) - (S) to (T) (64 Bits)
subuh	4	44	--------	Subtract; Upper result (R) - (S) to (T) (32 Bits)
subus	2	A4	**hllabsss**	Subtract; Upper result A - B ---> C
subuv	1	84	**hzoabsss**	Subtract; Upper result A - B ---> C
subx	4	67	--------	Subtract Address; (R) - (S) to (T)
subxv	1	87	**0zoab000**	Subtract Address; A - B ---> C
sum	1	DA	**hz000000**	Sum; (A0+A1+A2+ ...+n) To (C) and (C+1)
swcqta	7	18	--------	Shared Memory; CQTA to (T), (S) to CQTA
testio	7	1B	--------	Shared Memory; IQVF, Transfer Busy, Fatal Error
tfc	7	29	--------	Transmit Instrumentation Counter to (T)
tru	A	70	--------	Truncate; (R) to (T)
truh	A	50	--------	Truncate; (R) to (T)
truv	1	90	**hzoa0000**	Truncate; A ---> C

Table B-6. Instructions by Mnemonic (page 6 of 6).

Mnemonic	Format	Function	G-bits	Operation
truh	A	50	--------	Truncate; (R) to (T)
truv	1	90	**hzoa0000**	Truncate; A ---> C
vrevv	1	B8	**hzo00000**	Transmit Reverse; A ---> C
vsb	4	05	--------	Void Stack and Branch
vtov	1	98	**hzoa0000**	Transmit; A ---> C
vtovx	1	B7	**h000bfgr**	Scatter ---> Indexed C
vxtov	1	BA	**h0000fgr**	Gather ---> C
wait	D	FB	**00ss00pp**	Wait on Semaphore
wjtime	A	3A	--------	Transmit (R) to Job Interval Timer
xor	3	F0	--------	Logical Exclusive OR; A, B ---> C
xorn	3	F7	--------	Logical Exclusive OR NOT; A,B ---> C
xornv	1	9D	**hzoabnnn**	Logical Exclusive OR NOT; A, B, ---> C
xorv	1	9D	**hzoabnnn**	Logical Exclusive OR; A, B, ---> C

Appendix C: Instructions With Sign Control

Table C-1 lists the instruction operation codes for which sign control is valid. Each table entry shows the permitted values for G-bits 5, 6, and 7 of an instruction word.

Table C-1. Instructions for which sign control is valid.

Operation code	Function	G-Bits 5	6	7
80,81,82	Vector Add	0,1	0,1	0,1
84,85,86	Vector Subtract	0,1	0,1	0,1
88,89,8B	Vector Multiply	0,1	0,1	0,1
8C,8F	Vector Divide	0,1	0,1	0,1
93	Vector Square Root	0,1	0,1	0
A0,A1,A2	Sparse Vector Add	0,1	0,1	0,1
A4,A5,A6	Sparse Vector Subtract	0,1	0,1	0,1
A8,A9,AB	Sparse Vector Multiply	0,1	0,1	0,1
AC,AF	Sparse Vector Divide	0,1	0,1	0,1
CF	Arithmetic Compress	0,1	0,1	0,1
D8	Maximum of A -> C	0,1	0	0
D9	Minimum of A -> C	0,1	0	0

Appendix D: Instructions With Broadcasting

Table D-1 lists instructions that allow broadcasting of their A or B operands. Instructions are listed by their operation codes. Each table entry indicates whether A, B, or both, can be broadcast.

Table D-1. Instructions Allowing Broadcasting.

Operation code	Broadcast A	Broadcast B
80,81,82,83,84, 85,86,87,88,89, 8A,8B,8C,8F	Yes	Yes
90,91,92,93	Yes	No
94,95	Yes	Yes
96,97,98,99,9A	Yes	No
9B	Yes	Yes
9C	Yes	No
9D	Yes	Yes
A0,A1,A2,A4,A5 A6,A8,A9,AB,AC AF	Yes	Yes
B7	No	Yes
BB,BD	Yes	Yes
C0,C1,C2,C3,C4, C5,C6,C7	Yes	Yes
CF	No	Yes
D0,D4,DC	Yes	Yes

Appendix E: Instruction Termination Rules

The following tables list instructions (by operation code) with their terminating conditions. There are different tables for instructions that have different fields. Some abbreviations are used. They are:

- M-zero: Machine zero
- N-one: Normalized one
- I: Input
- O: Output

Table E-1. Instruction Terminating Conditions (part 1 of 6).

Instruction Code	A FIELD (INPUT)			C FIELD (OUTPUT)	
	A field exhausted	Extension type	Initial length zero	C field exhausted	Initial length zero
F8	Extend A	B designator byte	Extend A	Terminate	No-op

Table E-2. Instruction Terminating Conditions (part 2 of 6).

Instruction Code(s)	A FIELD (INPUT)			B FIELD (INPUT)			C FIELD (OUTPUT)	
	A field exhausted	Extension Type	Initial length zero	B field exhausted	Extension Type	Initial length zero	C field exhausted	Initial length zero
F0,F1,F2, F3,F4,F5, F6,F7	Extend A	Zero bits	Extend A	Extend B	Zero bits	Extend B	Terminate	No-op

Table E-3. Instruction Terminating Conditions (part 3 of 6).

Instruction Code(s)	A FIELD (INPUT)			B FIELD (INPUT)			C FIELD (OUTPUT)	
	A or X field exhausted	Extension Type	A or X length initially zero	B or Y exhausted	Extension Type	B or Y length initially zero	C or Z field exhausted	C or Z length initially zero
A0,A1,A2, A4,A5,A6, A8,A9,AB, AC,AF	NA	NA	NA	NA	NA	NA	NA	NA
	X FIELD (INPUT)			Y FIELD (INPUT)			Z FIELD (OUTPUT)	
	Extend X	Zero bits	Extend X	Extend Y	Zero bits	Extend Y	Terminate	No-op

Table E-4. Instruction Terminating Conditions (part 4 of 6).

Instruction Code(s)	A FIELD (INPUT)			B FIELD (INPUT)			C FIELD (OUTPUT)		
	A field exhausted	Extension Type	Initial length zero	B field exhausted	Exten. Type	Initial length zero	C field exhausted	Initial length zero	Control vector
80,81,82, 83,84,85, 86,87,8A	Extend A	M-zero	Extend A	Extend B	M-zero	Extend B	Terminate	No-op	Yes
88,89,8B, 8C,8F	Extend A	N-one	Extend A	Extend B	N-one	Extend B	Terminate	No-op	Yes
90,91,92, 93	Extend A	M-zero	Extend A	NA	NA	NA	Terminate	No-op	Yes
94,95	Extend A	M-zero	Extend A	Extend B	M-zero	Extend B	Terminate	No-op	Yes
96,97,98, 99,9A	Extend A	M-zero	Extend A	NA	NA	NA	Terminate	No-op	Yes
9B,9D	Extend A	M-zero	Extend A	Extend B	M-zero	Extend B	Terminate	No-op	Yes
9C	Extend A	M-zero	Extend A	NA	NA	NA	Terminate	No-op	Yes
B7	Terminate	NA	No-op	NA	NA	NA	NA	NA**	No
B8	Extend A	M-zero	Extend A	NA	NA	NA	Terminate	No-op	Yes (O)
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 A	M-zero	Extend A	Extend B	M-zero	Extend B	Terminate	No-op	Yes (O)
D1,D5	Extend A	M-zero	Extend A	NA	NA	NA	Terminate	No-op	Yes (O)
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)
DF	NA	NA	NA	NA	NA	NA	Terminate	No-op	Yes (O)

* These instructions may terminate even if the field length is not exhausted.
** These multiple pass instructions no-op for a group length equal to zero. Each pass of a multipass instruction terminates when this length equals zero.

Table E-5. Instruction Terminating Conditions (part 5 of 6).

Instruction Code(s)	R FIELD (INPUT)		S FIELD (INPUT)		T FIELD (OUTPUT)	
	R field exhausted	Initial length zero	S field exhausted	Initial length zero	T field exhausted	Initial length 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
1C,1D	Exit loop	String of all 0's or 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
70	Terminate data transfer to register file	No data transfer to register file	NA	NA	Terminate data transfer to register file	No data transfer to register file

* These instructions may terminate even if the field length is not exhausted

Table E-6. Instruction Terminating Conditions (part 6 of 6).

Instruction Code(s)	A FIELD (INPUT)			B FIELD (INPUT)			Z FIELD (INPUT or OUTPUT)		
	A field exh.	Extension Type	Initial length zero	B field exhausted	Extension Type	Initial length zero	Z field exhausted	Initial length zero	Control vector
BB,BC,BD	NA	NA	NA	NA	NA	NA	Term (I)	No-op (I)	No
C4,C5,C6 C7	Extend	M-zero	Extend	Extend	M-zero	Extend	Term (O)	No-op (O)	No
C8,C9,CA CB	Term.	NA	No-op	Exit search iteration	NA	Exit search iteration	NA	NA	Yes (O)
CC	Term.	NA	No-op	NA	NA	NA	NA	NA	No
CF	Term.	NA	No-op	Extend	M-zero	Extend	NA	NA	No
D8,D9	Term.	NA	No-op	NA	NA	NA	NA	NA	Yes (I)

Appendix F: Floating-Point Operations

Arithmetic on the ETA10 uses two's complement, floating-point procedures, allowing the computer to represent numbers with variable radix points. The computer automatically places the radix point of a result at the proper position following a computation. By shifting the radix point and increasing or decreasing the exponent, the machine can perform computations on widely varying quantities.

Floating-Point Format

Floating-point operations are performed on 32-bit and 64-bit operands. Floating-point numbers are expressed in scientific notation; a coefficient multiplied by an exponent (a number raised to a power), or $(2^x) \bullet c$, where c is the 24- or 48-bit signed coefficient, x is the 8- or 16-bit signed exponent, and the base is 2. Both exponent and coefficient are expressed as two's complement signed integers.

Figure F-1 shows a 32-bit floating-point number. Coefficients for 32-bit numbers range from -8,388,608 to +8,388,607 (#800000 to #7FFFFF). Exponents range from -112 to +111 (#90 to #6F). The exponent values from #8F to #70 fall into a special end-case range. Exponent values of *#8XXXXXXX* (where *X* equals any hexidecimal digit) represent machine zero. Exponent values of *#7XXXXXXX* (where *X* equals any hexidecimal digit) represent indefinite results. The minimum and maximum 32-bit values are approximately -2.177807E40 and 2.177807E40, with 7 or 8 digits of accuracy depending on the size of the number.

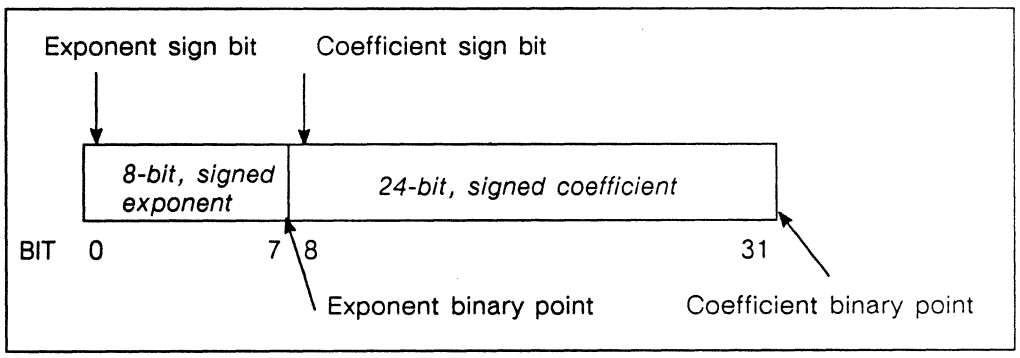


Figure F-1. A 32-Bit Floating-Point Number's Format.

Figure F–2 shows a 64-bit floating-point number. Coefficients range from –140,737,488,355,328 to +140,737,488,355,327 (#8000 0000 0000 to #7FFF FFFF FFFF), and exponents range from –28672 to +28671 (#9000 to #6FFF). Exponent values of #8*XXXXXXXXXXXXXXX* (where *X* equals any hexidecimal digit) represent machine zero. Exponent values of #7*XXXXXXXXXXXXXXX* (where *X* equals any hexidecimal digit) represent indefinite results. The minimum and maximum 64–bit values are approximately –9.53E8644 and 9.53E8644, with 14 or 15 digits of accuracy depending on the size of the number.

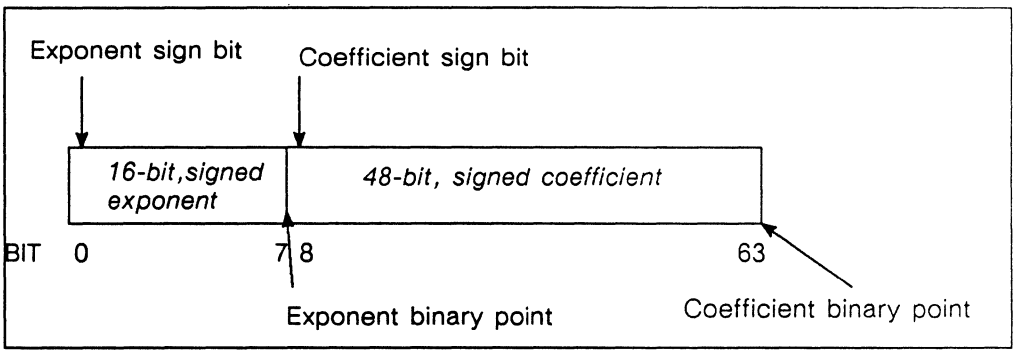


Figure F–2. A 64-Bit Floating-Point Number's Format.

Two's Complement Notation

In two's complement notation, the leftmost bit of the exponent and the leftmost bit of the coefficient are sign bits (zero is a positive sign bit and one is a negative sign bit). The remaining bits hold the numbers themselves.

In two's complement notation, positive number have the same representation they have in unsigned binary. For example, 4_{10} is equal to 0100; 9_{10} is equal to 01001; and so on. Note, however, that the sign bit must be 0 to indicate its positive value. If you place too large a positive value in the exponent or coefficient field, it will be interpreted as a negative number.

Negative number in two's complement notation are represented as a *complement* of their positive values. Representation of a negative value in two's complement notation is a simple two–step procedure. First, all ones are replaced by zeros, and all zeros are replaced by ones. Then, a one is added to the result. If a carry occurs from the left–most bit, it is thrown away.

To find the two's complement of the number -2965_{10}:

Begin with the binary equivalent of $+2965_{10}$ = 0101110010101

Now, replace all ones by zeros and all zero by ones =

1010001101010

Add one to the result = 1010001101011

And you find the two's complement notation of:
-2965_{10} = 1010001101011.

Another way of understanding two's complement notation is to understand that a number in two's complement notation is one more than the corresponding one's complement notation for the same number. For example, in two's complement, -1 is equal to #FFFFFF (all ones), while in one's complement, -1 is #FFFFFE. Positive numbers in two's complement are identical to the corresponding one's complement notation for the same number.

For an n-bit number,

positive numbers (k),

$0 \le k < 2^{n-1}$ using binary representation

negative numbers (k′),

$-(2^{n-1}) \le k' < 0$ represented by $2^n - k$, in binary representation

For example: when n=4,

if k = +5, then it is represented by 0101
if k = -5, then it is represented by 1011.

So, if the binary representation is

$$a_0 a_1 a_2 \ldots a_{n-1}$$

the value is $a_0(-2^{n-1}) + \sum_{i=1}^{n-1} a_i(2^{n-1-i})$.

Note that the sign bit (a_0) has negative weight and all other bits have positive weight.

Floating-Point Arithmetic

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 (left) and lower (right) result. Figure A–3 shows their format.

The sign bit of the lower result's coefficient is not affected in a lower operation. It remains zero in two's complement arithmetic; the other bits of the lower coefficient receive no such special treatment.

A lower result is not meaningful alone, but must be used in conjunction with its associated upper result. Data flags resulting from the lower result pertain only to the lower result.

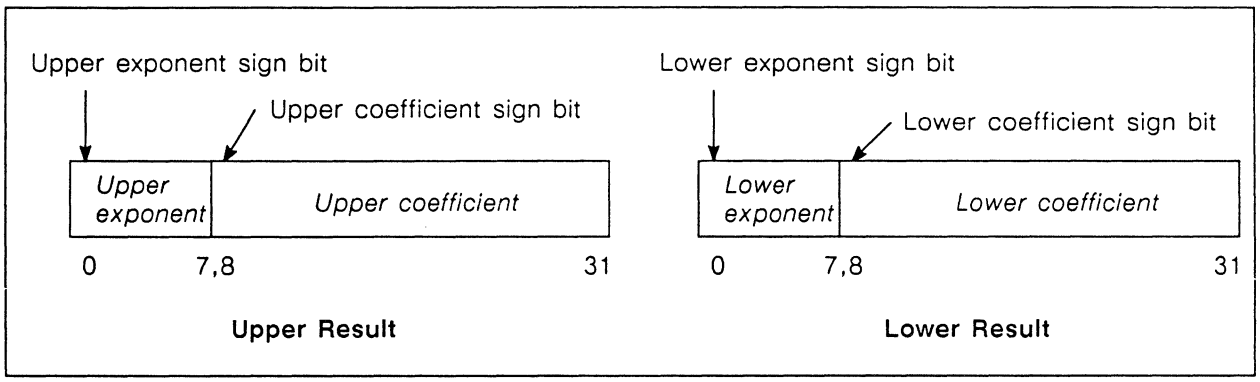


Figure A–3. Floating-Point Result Formats for Add, Subtract, and Multiply Operations.

Right Normalization

Right normalization is performed in the ETA10 when the result coefficient overflows its register. When this happens, the entire result is shifted right one place. The sign bit is extended, and the exponent is increased by one.

Right normalization is performed when necessary, regardless of whether the instruction specifies normalization. If right normalization causes an exponent overflow, the result is set to indefinite and data flag bit 42 is set.

Floating-Point Addition

Before addition takes place, both operands' coefficients are extended to 94 bits for 64-bit operands and 46 bits for 32-bit operands (not including a sign bit) by adding zeros to the right of the operands binary point, see figure A-4.

The exponents of the two operands are then compared. The coefficient of the operand with the smaller exponent is shifted right one bit and its exponent increased by one, successively, until the operand's exponents are equal. The shifted coefficient's sign is extended from left to right during the shift. Negative coefficients approach a minus one, and positive coefficients approach zero as they are shifted.

The addition is a 94-bit (46 for 32-bit operands) conventional binary addition. Right normalization takes place if necessary. The coefficient for the upper result is the left-most 47 bits (23 bits for 32-bit operands), excluding the sign bit. The coefficient for the lower result is the right-most 47 bits (23 bits for 32-bit operands) of the 94-bit (46-bit for 32-bit operands) result.

The exponent of the upper result is the larger of the source exponents. If right normalization occurred, the value is increased by one.

The lower result's exponent is 47 (23 for 32-bit operands) less than the upper result's exponent for all cases except when:

- Right-normalization causes the upper result exponent to overflow. The upper result is set to indefinite. In this case, the lower exponent is #6FD1 (#59 in the 32-bit case).
- The upper result's exponent minus 47 (23 for 32-bit operands) causes exponent overflow. In this case, the lower result is set to machine zero.
- Either or both operands were indefinite. In this case, the upper and lower results are indefinite.

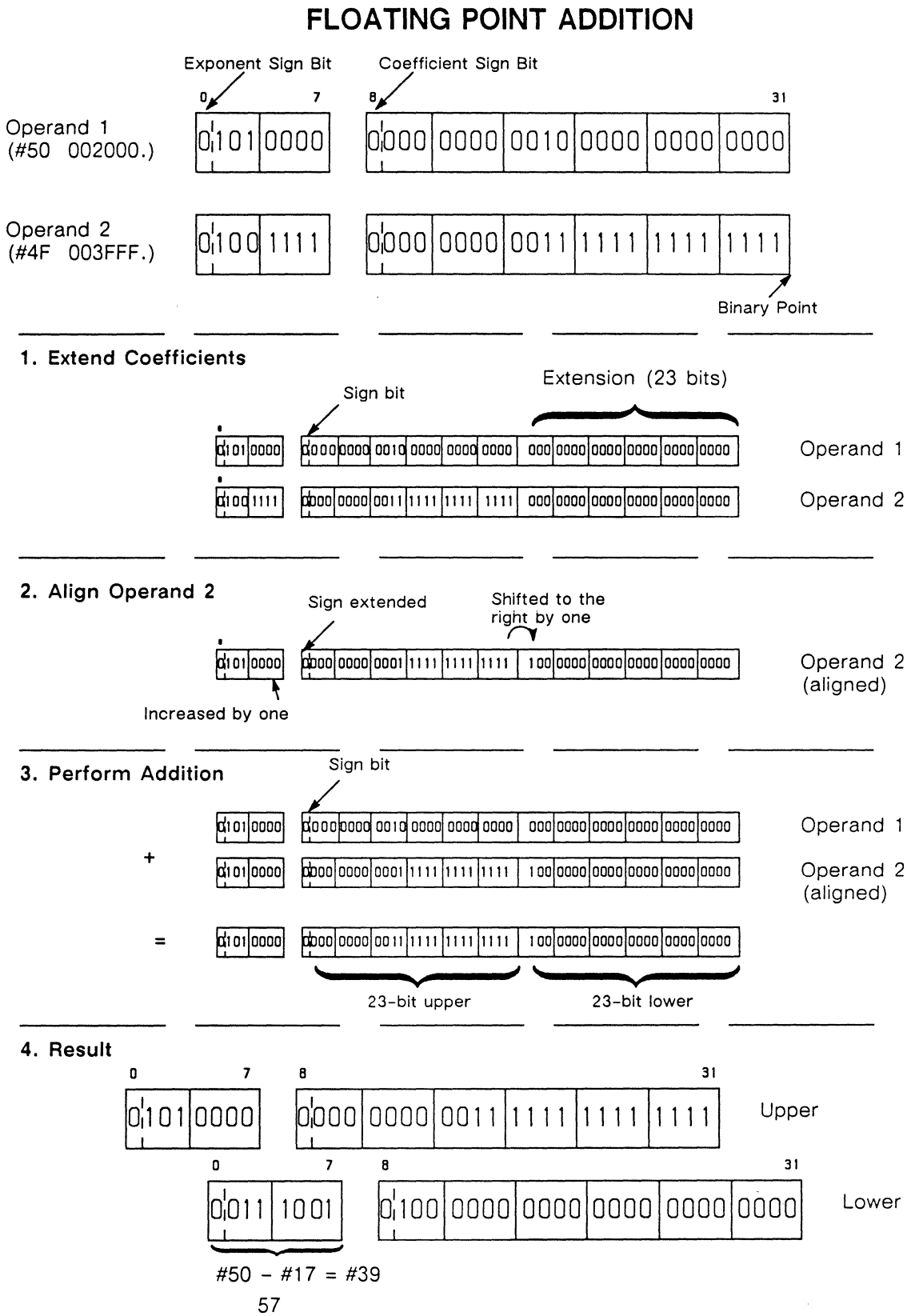


Figure A-4. Floating-Point Addition.

Floating-Point Subtraction

Floating-point subtraction is performed by complementing the coefficient of the subtrahend, and then performing a floating-point addition, refer to figures A–4 and A–5.

The complement is a 48-bit (24-bit for 32-bit operands) two's complement operation is performed before the operands are extended to 94 bits (46 bits for 32-bit operands).

Note: 1. The complement of a coefficient of #8000 0000 0000 (#80000 for 32-bit operands) is #4000 0000 0000 (#40000 for 32-bit operands). One is also added to the exponent.

2. A subtract operation is not always commutative. For example, it is not true that A–B = –(B–A) when:

- The exponents of A and B are not equal.
- '1' bits exist in any of the right-most bit positions of the coefficient that will be shifted off to the right during alignment of the smaller exponent.

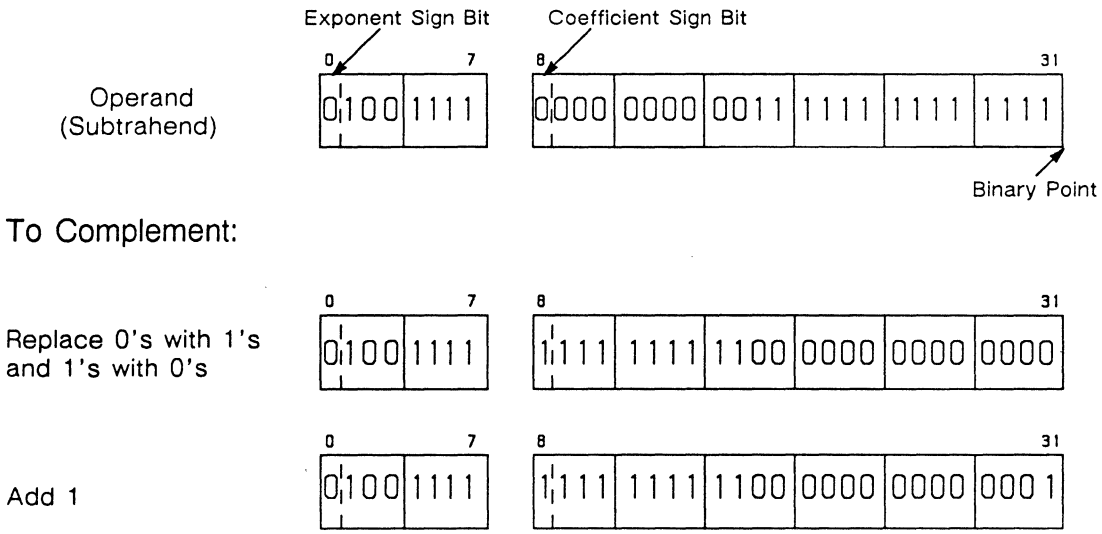


Figure A-5. To perform floating point subtraction, complement the subtrahend, then add.

Floating-Point Multiplication

When two 64-bit floating-point numbers are multiplied, the 47 least significant product bits that are generated are placed in the lower result, and the higher order 47 bits in the upper result. For 32-bit operands, only 23 bits go into the upper and lower result. See figure A-6.

The sign bit of the lower result is always cleared to zero, and the exponent of the lower result is the sum of the two source operands' exponents, except as listed below.

The sign of the upper result's coefficient follows the normal rules of algebra. The upper result's exponent is the sum of the two source exponents plus 47 (23), except as listed below.

Exceptions:

- The sum of the source operands' exponents, plus 47 (23 for 32-bit operands) if upper result, exceeds #6FFF (#6F for 32-bit operands). The result exponent is set to indefinite.
- The sum of the source operand's exponents (plus 47 (23 for 32-bit operands) if upper result) is less than #9000 (#90 for 32-bit operands). The result exponent is set to machine zero.
- Either or both operands are indefinite. The result exponent is set to indefinite.
- Neither operand is indefinite, but either or both are machine zero. 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), and an exponent of x, the operand is treated as if its coefficient were #C000 0000 0000 (#C00000 for 32-bit operands), and its exponent were x+1.

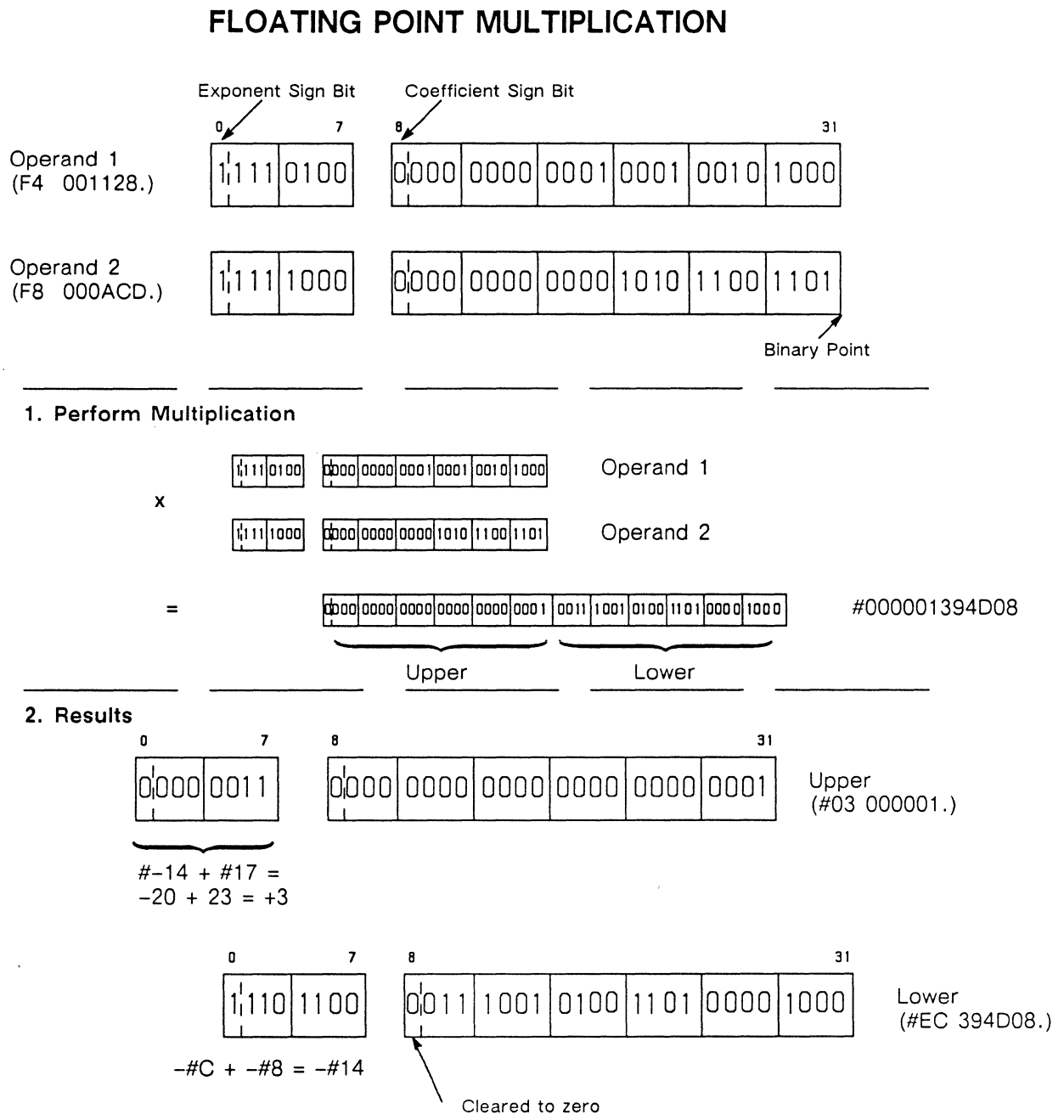


Figure A-6. Floating-Point Multiplication.

Floating-Point Division

The division operation, figure A–7, divides the pre-normalized coefficient of the divisor into the dividend's coefficient. A 48-bit (24-bit for 32-bit operands) quotient is generated as the upper result.

Except for the calculation of significance, if either operand has a coefficient of #8000 0000 0000 (#800000 for 32-bit operands), the operand is treated as if its coefficient were #C000 0000 0000 (#C00000 for 32-bit operands), and its exponent increased by one.

When the divide hardware normalizes the divisor's coefficient, the number of places shifted to the left is added to the quotient's exponent according to the following equation.

Exponent of Quotient =

(dividend's exponent) – (divisor's exponent) – *(constant – N)*

Where *constant* is 46 (22 for 32-bit operands), and *N* is the number of places shifted left to pre-normalize the divisor.

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

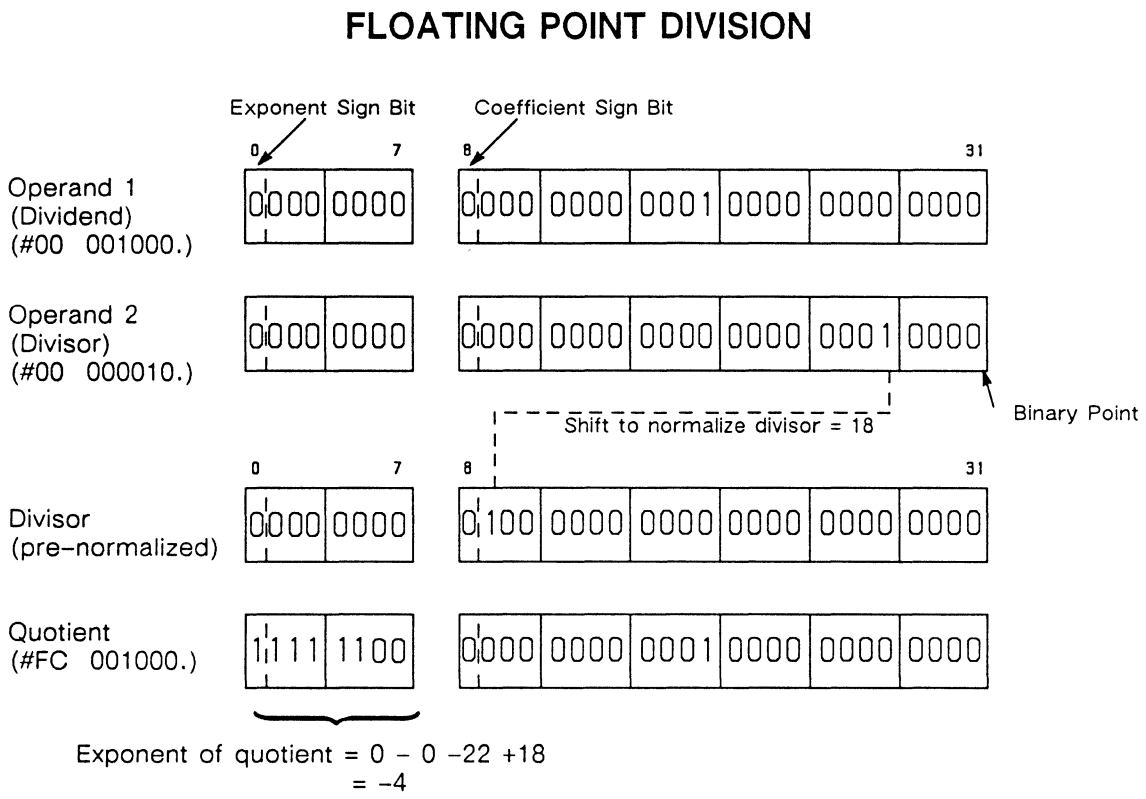


Figure A-7. Floating-Point Division.

Normalized Upper Results

The normalized add and subtract instructions (i.e., #42 and #46) generate an intermediate result that is identical to the final result of the Add U and Subtract U instructions (for example #40 and #44), except that an operand with a coefficient of all zeros is treated as machine zero.

A floating-point number is normalized by left-shifting the coefficient until the sign bit does not equal the next bit to the right. (This implies that the coefficient has been shifted to the left as far as possible.)

During the shift, zeros are attached to the right end of the coefficient, and the exponent is reduced by one for each left shift. If reducing the exponent by one causes exponent underflow, the result of the normalization operation is defined as machine zero.

Note: Normalization of an all-zero coefficient results in machine zero.

Double-Precision Results

Some instructions (such as #DA and #DC) produce double-precision results. A double-precision floating-point add operation is nothing more than a floating–point add that produces an upper and lower result simultaneously and retains both results for the next floating-point operation. The partial result consists of 94 coefficient bits plus sign information (for 64-bit operands), and 46 bits plus sign information (for 32-bit operands).

Dot Product instructions add both the upper and lower results of the multiply instructions to the partial results of add operations, as described above.

Floating-Point Square Root Operations

The ETA10 performs floating-point square root operations in the following steps:

1. Determine and record the significance of the input operand's coefficient.
2. If the significance is negative, complement the input operand to its positive form.
3. If the exponent of the input operand is odd, reduce it by one, and multiply the coefficient obtained in step 2 by two. If the exponent is even, do not modify it.
4. Obtain the coefficient's square root from step 3. Attach enough zeros to the right end of the coefficient to produce 48 (24 for 32-bit operations) result bits.
5. If the original input operand was negative, complement the result coefficient. If the original input operand was positive, do not modify the result.
6. Form the result exponent by dividing the exponent (obtained in step 3) by two, and subtracting 23 (11 for 32-bit operands).
7. Adjust the result coefficient to produce a coefficient with the same significance as the input operand, using the significance count obtained in step 1. Adjust the result's exponent to compensate for the result coefficient's change in magnitude.

An input operand with a zero coefficient produces 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 input operand is negative, data flag bit 45 is set. If it is indefinite or machine zero, the result is indefinite or machine zero respectively, and data flag bit 45 is not set.

Except for the calculation of significance, if either operand has a coefficient of #8000 0000 0000 (#800000 for 32-bit operands), the operand is treated as if its coefficient were #C000 0000 0000 (#C00000 for 32-bit operands), and its exponent is increased by one.

Significant Results

Certain multiply, divide, and square root instructions generate significant results for the product or quotient.

A floating-point number's significant bit count equals 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 all-one coefficient has a significant bit count of zero.

Note: A positive non-zero coefficient that is an exact power of two has a significant bit count that is one greater than its negative form. An input operand's significance is determined from the operand as originally read from a register or central memory before performing any operation such as sign control, handling a coefficient of #8000 0000 0000 (#800000 for 32-bit operations), or performing a left shift for odd exponents in a square root operation.

Significant arithmetic determines which source operand has the smaller significant bit count, and records the count. After the arithmetic operation, the result's significant bit count is determined after any sign correction takes place. The significant bit counts of the input and the result are compared.

If the result's significant bit count is less than the input's significant bit count, the result coefficient is left shifted (with zeros shifted in) by the difference, and the exponent reduced accordingly.

If the significant bit counts are equal, the coefficient is not shifted, nor is the exponent adjusted.

If the result's significant bit count exceeds the input's, the result coefficient is right-shifted (end-off with sign extension) and the exponent increased accordingly.

Note: For a multiply operation, the entire 95-bit result (47-bit 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 can take a result out of exponent overflow or underflow.

Floating-Point Comparison Rules

Some instructions compare two floating-point operands (r and s) for:

- Equality ($r = s$)
- Non-equality (r not $= s$)
- Greater than or equal ($r >$ or $= s$)
- Less than ($r < s$)

Certain floating-point comparison rules apply, depending on the operands.

Indefinite Operand(s)

If one of the operands is indefinite, the comparison is not met, because by definition an indefinite number is not greater than, less than, equal to, or not equal to, any other operand.

If both operands are indefinite, the ($r = s$) and ($r >$ or $= s$) conditions are met, because an indefinite number is defined as being equal to another indefinite number.

Machine Zero Operand(s), Not Indefinite

An operand that is not indefinite, not machine zero, and has a positive, non-zero coefficient, is greater than machine zero.

An operand that is not indefinite, not machine zero, and has a negative coefficient, is less than machine zero.

Machine zero is equal to itself and to any number with an exponent that is not indefinite and has an all-zero coefficient.

Operand(s) Not Indefinite or Machine Zero

Operands are unequal if their coefficients have unlike signs. The operand with the positive coefficient is the greater.

If the operands' coefficients have like signs, a floating-point subtract Upper operation ($r - s$) is performed on them to compare the two operands.

- If the upper 48 bits (24 for 32-bit operations) of the result coefficient are all zeros, then $r = s$.
- If the upper 48 bits (24 for 32-bit operations) of the result coefficient are not all zeros, then r *not* $= s$.
- If the result coefficient is positive, then $r >$ or $= s$.
- If the result coefficient is negative, then $r < s$.

There is no guarantee that if $r = s$, $s = r$ under the following conditions (these conditions can exist only if the operands are not normalized):

- The operands have unequal exponents.
- '1' bits exist in any of the rightmost bit positions of the coefficient. They will be shifted off the right during alignment of the smaller exponent.

The following example shows how $r - s$ is not equal to $s - r$.

Assume *Operand* r= #0100 0000 0000 1001 *and*
Operand s= #0104 0000 0000 0100

Complement *s:* #0104 FFFF FFFF FF00
and align *r:* #0104 0000 0000 0100 1

Add *r* and *s:* #0104 0000 0000 0000 1

Since the upper 48 bits of the result's coefficient are zeros, $r = s$. However, if the operands are interchanged, the result is different.

r: #0104 0000 0000 0100
s: #0100 0000 0000 1001

Complement *s:* #0100 FFFF FFFFEFFF
and align *r:* #0104 FFFF FFFFFEFF F

Add *s* and *r:* #0104 FFFF FFFFFFFF F

Since the upper 48 bits of the result coefficient are not all zeros, the operands *s* and *r* are not equal.

Appendix G: The Data Flag Register

The data flag register provides an automatic branch to a special routine for certain operands, results, or conditions, without incurring the penalty of explicit program checking for those conditions. If a condition previously selected to cause an automatic branch occurs during an instruction, the address of the next instruction that would have been executed is stored in the address portion of register 01, and a branch made to the address in register 02. Zero, one, or more instructions may be executed before an automatic branch actually occurs.

The data flag register is located in word 4 of the Invisible Package, Domain Package, and the Stacked Domain Package.

Data Flag Register Format

Figure G-1 shows the data flag register. Bits 0-2, 16-18, 32-34, and 48-50 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. Fields in the data flag register are explained in the following sections.

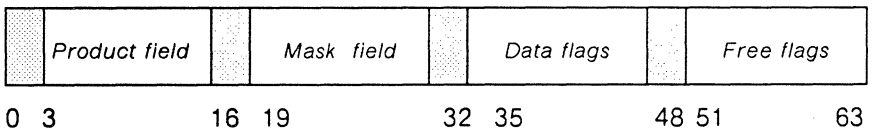


Figure G-1. Data Flag Register Format.

Data Flags

Data flags in bits 35–47 indicate conditions that have occurred. For example, bit 37 is set at the end of a #CC instruction (Masked Binary Compare) if no match is found. Another #CC instruction that finds a match will not clear bit 47. The only instructions that will clear the data flag bits are #33 (Data Flag Register Bit Branch and Alter) and #3B (Data Flag Register Load/Store). A Job to Monitor exchange also clears the data flag register.

If a control vector is used in a vector operation, the current control vector bit must be permissive in order to set any of data flag bits 41–46. If a divide fault occurs, but the control vector bit for the result element is not permissive, the divide fault data flag is not set.

The Mask Field

Each data flag is associated with a mask bit that selects the conditions for which a programmer wants an automatic data flag branch.

The associated mask bit need not be set in order to set a data flag bit. The mask function simply enables a particular data flag to cause a bit to be set in the product field. The order in which a mask bit and its associated data flag bit are set is immaterial, as the result is the same; their associated product bit is set.

Product Field Bits

Each product bit is the dynamic logical product of a data flag bit and its associated mask bit. A data flag branch may occur when at least one bit is set in the product field.

Data Flag Branch Enable Bit

Bit 52, the data flag branch enable bit, must be set for a branch to occur. The hardware clears bit 52 automatically when a branch takes place. Bit 52 must be reset with a #33 (Data Flag Register Bit Branch and Alter) or a #3B (Data Flag Register Load/Store) instruction.

Causing a Data Flag Branch

If a mask field bit and its associated masked data flag bit are set, the associated product field bit is also set. Bit 51 in the free flag field also becomes a one, since it is the dynamic inclusive OR of bits 3–15 of the product field.

If bits 51 and 52 are set, an automatic data flag branch (DFB) occurs after termination of the instruction that caused the DFB. The next instruction's bit address is loaded into the right-most 48 bits of register 01, and control branches to the bit address in the right-most 48 bits of register 02. Bit 52 is automatically cleared. The left-most 16 bits of register 01 are cleared to zero. The address in register 01 points to an instruction that is zero or more instructions removed from the instruction that caused the DFB.

Note: When bit 52 is cleared, DFBs are disabled. However, 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 enter a tight loop. To prevent this situation for all cases except those involving the Job Interval Timer, bit 51 should be tested for a zero before setting bit 52.

When using the Job Interval Timer, bit 36 is set asynchronously with respect to instruction execution, once the Job Interval Timer is loaded. The timer may set bit 36 after the check of bit 51 and before the branch to the contents of register 01. One way to handle this is to examine register 01's contents upon entering the data flag branch routine. If register 01 indicates that the branch occurred outside the DFB routine, then register 01 can be copied to a temporary location. If the branch occurred within the temporary location, 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 address in register 01 into a single 33 instruction (Data Flag Register Bit Branch and Alter), whose instruction word is 33603401.

Data Flag Register Bit Assignments

Tables G–1 and G–2 list the data flag register product bit, mask bit, and data flag bit settings and their meanings.

Table G–1. Data Flag Bit Settings (page 1 of 2).

Product Bit	Mask Bit	Data Flag Bit	Meaning
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 Invisible Package. After exchanging back to Job mode, if bit 35 and its corresponding mask bit (bit 19) are set, a normal Data Flag branch occurs.
4	20	36	Job Interval Timer.
5	21	37	Select condition not met. Valid for instructions C0–C3, or if no match found on CC instruction.
6	22	38	Unused.
7	23	39	The binary result exceeds the range 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 is machine zero. If the divisor and/or the dividend is indefinite, there is no divide fault. If a divisor causes a divide fault, the quotient is set to indefinite. However, "exponent overflow" and "result machine zero" data faults are not set.
10	26	42	Exponent overflow. The result's exponent exceeds #6FFF (#6F for 32-bit arithmetic). Results are checked for exponent overflow after the exponent is adjusted for normalization or significance. In the adjust exponent instructions, this data flag is set if a left shift exceeds the number of places required for normalization. Exponent overflow causes an indefinite result, therefore the indefinite flag is always set on exponent overflow. The exponent overflow data flag is not set when either source operand is indefinite, or when the divisor on a divide instruction causes a divide fault.

Table G-2. Data Flag Bit Settings (page 2 of 2).

Product Bit	Mask Bit	Data Flag Bit	Meaning
11	27	43	Result machine zero. A result's exponent is less than #9000 (#90 for 32-bit arithmetic). Result machine zero may be caused by exponent underflow, or by a machine zero input operand. A divide instruction whose divisor causes a divide fault does not set the result machine zero data flag bit.
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 square root instruction has a negative source operand. The square root of the operand's absolute value is formed and its complement stored as the result.
14	30	46	An indefinite result was formed, or either or a floating-point compare operation had indefinite operand(s). An indefinite result may occur when one or both operands of a floating-point arithmetic operation are indefinite, or when a divide fault or exponent overflow occurs.
15	31	47	A breakpoint occurred.

Free Data Flags

Bit 51 is the dynamic inclusive OR of the product field. This bit is set if any of bits 3 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. Data flag branch execution automatically clears bit 52.

Bits 53, 54, and 55 have no associated product or mask bits. They are cleared by instructions that then may set any of them, unless the instruction is a no-op. If pertinent, these bits must be sampled before executing another instruction that would alter their previous state. Setting these bits does not cause a data flag branch. Table G-3 lists their meanings for different instructions.

Table G-3. Definitions For Free Data Flag Bits 53-55.

Instruction	Bit 53	Bit 54	Bit 55
F0-F7	Result field all zeros	Result field mixed	Result field all ones
1E	Ones were counted	Undefined	Undefined
D8,D9	Undefined	More than one element met criteria	Undefined
28	Whole field was scanned, no hit	Undefined	Undefined

Bits 56 through 63 have no associated product or mask bits. They help software determine the operation that caused bits 41, 42, 43, 45, and 46 to be set.

Bit 56 Unused.

Bit 57 Unused.

Bit 58 A scalar convert, divide, or square root operation set bits 39, 41, 42, 43, 45, and/or 46.

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

Bit 60 Vector pipes exponent overflow. Duplicate of bit 42, caused by a vector.

Bit 61 Vector pipes machine zero result. Duplicate of bit 43, caused by a vector.

Bit 62 Vector pipes square root result imaginary. Duplicate of bit 45, caused by a vector.

Bit 63 Vector pipes indefinite result. Duplicate of bit 46, caused by a vector.

Instructions Affecting Data Flag Register Bits

Table G-4 shows the data flag bits set by the instructions. An X indicates that the change is dependent on the data processed. An A indicates the data flag register is explicitly altered.

Table G-4. Data Flag Bits set by function codes. (Page 1 of 3)

Function Code	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										
1F										X
20								X		
21								X		
22								X		
23								X		
24								X		
25								X		
26								X		
27								X		
28										X
29										
2A										
2B										

Function Code	Data Flag Bits 37	38	39	41	42	43	45	46	47	53 54 55
2C										
2D										
2E										
2F										
30										
31										
32										
33	A	A	A	A	A	A	A	A	A	A
34										
35										
36										
37										
38										
39										
3A										
3B	A	A	A	A	A	A	A	A	A	A
3C										
3D										
3E										
3F										
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										

Table G-4. Data Flag Bits set by function codes. (Page 2 of 3)

Function Code	Data Flag Bits 37	38	39	41	42	43	45	46	47	53 54 55
58					X	X	X			
59										
5A										
5B										
5C					X		X			
5D					X		X			
5E										
5F										
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										
80					X	X		X		
81						X		X		
82					X	X		X		
83										

Function Code	Data Flag Bits 37	38	39	41	42	43	45	46	47	53 54 55
84					X	X		X		
85						X		X		
86					X	X		X		
87										
88					X	X		X		
89					X	X		X		
8A										
8B					X	X		X		
8C				X	X	X		X		
8D										
8E										
8F				X	X	X		X		
90								X		
91								X		
92								X		
93						X	X	X		
94					X	X		X		
95					X			X		
96					X	X		X		
97					X	X		X		
98										
99					X	X		X		
9A										
9B										
9C						X		X		
9D										
9E										
9F										
A0					X	X		X		
A1					X	X		X		
A2					X	X		X		
A3										
A4					X	X		X		
A5					X	X		X		
A6					X	X		X		
A7										
A8					X	X		X		
A9					X	X		X		
AA										
AB					X	X		X		
AC				X	X	X		X		
AD										
AE										
AF				X	X	X		X		

Table G-4. Data Flag Bits set by function codes. (Page 3 of 3)

Function Code	Data Flag Bits 37	38	39	41	42	43	45	46	47	53 54 55
B0								X		
B1								X		
B2								X		
B3								X		
B4								X		
B5								X		
B6										
B7										
B8										
B9										
BA										
BB										
BC										
BD										
BE										
BF										
C0	X							X		
C1	X							X		
C2	X							X		
C3	X							X		
C4								X		
C5								X		
C6								X		
C7								X		
C8								X		
C9								X		
CA								X		
CB								X		
CC	X									
CD										
CE										
CF								X		
D0						X		X		
D1						X		X		
D2										
D3										
D4						X		X		
D5					X	X		X		
D6										
D7										
D8								X		X
D9								X		X
DA					X	X		X		
DB					X	X		X		

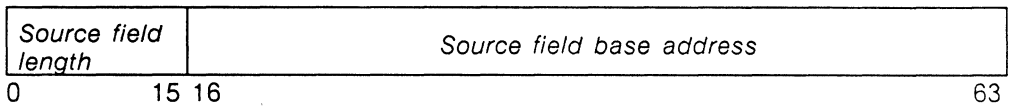
Function Code	Data Flag Bits 37	38	39	41	42	43	45	46	47	53 54 55
DC					X	X		X		
DD										
DE										
DF					X	X		X		
E0										
E1										
E2										
E3										
E4										
E5										
E6										
E7										
E8										
E9										
EA										
EB										
EC										
ED										
EE										
EF										
F0										X
F1										X
F2										X
F3										X
F4										X
F5										X
F6										X
F7										X
F8										
F9										
FA										
FB										
FC										
FD										
FE										
FF										

Appendix H: Addressing Vector Operands

Vector instructions perform operations on ordered scalars. Instruction designators point to registers describing sources and destinations. The sources and destinations are in memory, and are vectors rather than single quantities.

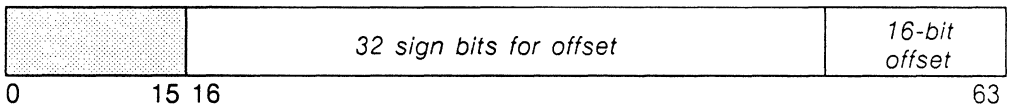
Addressing Vector Source Operands

The A and B instruction designators specify registers holding the base address and field length of the source operand fields A and B, giving the memory location of vector A and vector B. The format of the source register is:



Source Operand Offsets

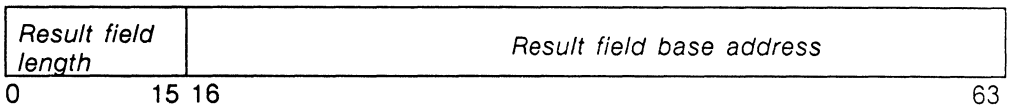
Designators X and Y specify registers that hold the offsets for the source operand fields A and B respectively. If the offset is over 16 bits long, the instruction is undefined. Bits 0–15 of the register are unused. The register's contents are:



A source vector's starting address is calculated by adding its base and offset. Prior to the addition, the offset, an item count, is shifted to the left five or six places to properly align it with the base address. The portion of the vector that will be included in the A or B vector stream for the instruction is calculated by subtracting the offset from the source field length. The resulting vector length must be greater than zero, and less than 2^{16}. A negative result is treated as a zero vector length.

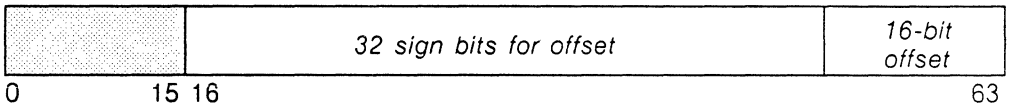
Addressing Vector Result Operands

The C instruction designator specifies the register holding the base address and field length of the result operand field C, giving the memory location of vector C. The format of the result register is:



Result Operand Offsets

If vector C has an offset (bit 2 of the G-field is set to one by the *z* qualifier), the C+1 register holds the offset. This offset also applies to a control vector, if used. If an offset applies, C should be an even numbered register, otherwise the instruction is undefined.



A result vector's starting address is calculated by adding its base and shifted offset. The portion of the vector included in the C vector stream for the instruction is calculated by subtracting the offset from the result field length. The resulting vector length must be greater than zero, and less than 2^{16}. A negative result is treated as a zero vector length. The control vector assumes the same length as the C vector stream.

Appendix I: Illegal Instructions

There are two types of illegal instructions: Type One illegal instructions and Type Two illegal instructions.

Type One Illegal Instructions

Type One illegal instructions include all unused function codes.

Instruction #7D becomes a Type One illegal instruction when its operands cause it to attempt to transfer a vector with an odd starting address (register file or CPM address) or an odd length.

Type One illegal instructions can occur in both Monitor and Job mode. If the illegal instruction occurs in Job mode, it sets bit 58 of the Interrupt Register (IR) to a one and then waits for an Interrupt Exchange. The non-zero IR causes an exchange to Monitor mode where execution starts at the address in Monitor's #03 register.

If the illegal instruction occurs in Monitor mode, it sets bit 58 of the Interrupt Register (IR) to a one and then waits for a Master Clear signal. There is no exchange, but execution begins at the address in Monitor's #03 register after the Master Clear. Bit 58 in the IR is supplied to the Maintenance Port as a CPU Status bit. The CPU should be able to execute Stop and S-REG operations, but a Master Clear may be needed to recover.

Type Two Illegal Instructions

Type Two illegal instructions include:

- Monitor mode instructions #00, #0A, and #0C through #0F that attempt to execute in Job mode.

- Any instruction that has a function code corresponding to a bit set to one in the Domain Package's Illegal Instruction Mask, (see Table A-1). Possible Type Two illegal instructions include: #07 through #09, #18 through #1B, #57, and #FA through #FF.

Table A-1. The Domain Package's Illegal Instruction Mask.

Bit Assignment	Function Code
0	Undefined, bit must be zero
1	Undefined, bit must be zero
2	Undefined, bit must be zero
3	07
4	08
5	09
6	Undefined, bit must be zero
7	Undefined, bit must be zero
8	Undefined, bit must be zero
9	Undefined, bit must be zero
10	18
11	19
12	1A
13	1B
14	Undefined, bit must be zero
15	57
16	Undefined, bit must be zero
17	Undefined, bit must be zero
18	Undefined, bit must be zero
19	Undefined, bit must be zero
20	Undefined, bit must be zero
21	Undefined, bit must be zero
22	Undefined, bit must be zero
23	Undefined, bit must be zero
24	Undefined, bit must be zero
25	Undefined, bit must be zero
26	FA
27	FB
28	FC
29	FD
30	FE
31	FF

The Domain Package's Illegal Instruction Mask is a 32-bit number that allows a particular instruction or group of instructions to be selected as legal or illegal instructions for different Domain packages. It is defined in the invisible package and the Domain package. Instructions are legal if their corresponding bit is set to zero; they are illegal if the bit is set to one. The Illegal Instruction Mask has no effect in Monitor mode.

In addition to the illegal instructions noted above, the #36 instruction also becomes a Type Two illegal instruction under the following two conditions:

- If its operands cause it to attempt a forward Domain change to a new Domain Package that corresponds to a bit cleared to zero in the Forward Domain Change Mask of the current Domain package.
- If its operands cause it to attempt a forward Domain change when the current value in the Stack Index Register is not less than the value in the Stack Limit Register.

Type Two illegal instructions can occur only in Job mode. When a Type Two illegal instruction occurs, it sets bit 59 of the Interrupt Register (IR) to a one and then waits for the non-zero IR to cause an exchange to Monitor mode. Execution then begins at the address in Monitor's #03 register.

Glossary

BCD	Binary Coded Decimal
BLAP	Base/Limit/Access Pair
CB	Communication Buffer
CPU	Central Processing Unit
CQTA	Completion Queue Tail Address
DFB	Data Flag Branch
IOU	Input-Output Unit
IQHA	Input Queue Head Address
IQTA	Input Queue Tail Address
IQVF	Input Queue Valid Flag
SM	Shared Memory
SU	Service Unit
TRB	Transfer Request Block
TRBSA	Transfer Request Block Store Address

Access Interrupt — Any addressing of storage that is not in CP memory, or any addressing that attempts an access in violation of the storage's allowed access.

Associative Registers — The set of 16 registers in the associative unit in which the space table is rippled through and read until a match for the requested virtual address is made. These registers perform the virtual-to-physical translation of page addresses.

Associative Word — Contains the virtual and physical address of each page in central processor memory; these words are read by the associative registers to do virtual-to-physical address translation.

Base/Limit Access Pair (BLAP) — Two memory words that reside in the domain package and denote the lowest communication buffer address the domain can access (base), the highest address the domain can access (limit), and the access rights the domain has to those addresses (access).

Binary Coded Decimal — A number with 15 digits; there are four bits per digit, plus the sign in the lower bits (60–63).

Breakpoint Register A maintenance and program debugging aid. Contains a breakpoint address and function for CPU write operands and/or CPU read operands. The breakpoint function compares addresses of specific categories of requests with the breakpoint address.

Broadcast constant A constant that becomes a source operand in vector operations. Used for each element of the vector stream for the length of the operation.

Central Processing Unit The combination of central processor and CP memory that is a computational engine in the system.

Central Processor A scalar processor, vector processor, file registers and interfaces to other system components.

Communication buffer Memory accessible from all central processors in the system, used for transmission of high-speed messages and signals among the system components.

Control Vector A bit vector. The setting of each control vector bit determines whether a result element is stored in the corresponding output vector field, and whether data flag bits are set.

CP memory A CPU's memory, accessible strictly from its associated central processor (and so to shared memory), and from the Service Unit.

Data Flag Branch Register A 64-bit register containing data that enables programs to branch to special routines when certain conditions or results occur.

Designator A bit group that makes up a field in an instruction word format. Usually 8 bits long. Each field is represented by a letter indicating its type; for example, F is the function.

Domain A CPU hardware feature used to define the CP memory access keys, the CB base/limit access pairs, and domain change information for controlling a process.

Domain change A generic name given to related operations that are the result of executing the #36 and #17 instructions. Causes a branch from one job code to another.

Domain Package A package loaded by the forward domain change instruction (#17) during a domain change.

Double precision Result The result produced by instructions such as #DA and #DC, that perform a floating-point addition that produces an upper and lower result simultaneously. Both results are retained for the next floating-point operation.

Exchange A central processor switch between monitor mode and job mode; exchanges are caused by a hardware interrupt or by an #09 instruction.

Exchange to Job Mode Puts the central processor into job mode to start a new process or to resume execution of an interrupted process.

Exchange to Monitor Mode Puts the central processor into monitor mode to do cleanup required by a completing process or to respond to an interrupt received by the processor.

External interrupt

Floating-point number A 64- or 32-bit number containing an exponent and coefficient expressed as two's complement, signed integers of the form $(2^x)*C$, where C is a signed coefficient, x is the exponent, and the base is two.

Full Word A 64-bit quantity in which the address of the left-most bit is a multiple of 64. The lowest six bits of a full word address are set to zero.

G-bit One of 8 bits in the G designator (subfunction field) of an instruction word.

Half Word A 32-bit quantity in which the address of the left-most bit is a multiple of 32. The lowest five bits of a half word address are set to zero.

Invisible package A package that defines and saves many significant characteristics controlling the instructions executed during an exchange.

Internal interrupt

Interrupt register Contains bits that, if set to one since the last #0E instruction was executed, cause an exchange to monitor mode.

Instruction Set The ETA10 set has 256 function codes, 40 of which are unused; it is model independent and vector-oriented. The set is compatible with Control Data Corporation CYBER 205 instructions.

Item count A field length, offset, index, or shift count that specifies a number of bits, bytes, half words, or full words.

Job A collection of commands that is scheduled, executed, and thought of as a unit. May execute through batch or interactive sessions. Also defined as a session; the basic unit of work on the ETA10.

Job Interval Timer A 32-bit timer used by application programs to time execution intervals.

Job Mode The period in central processor operations during which the processor fetches, executes, and returns results from instructions contained in user programs.

Lower result The right half of a result coefficient generated by floating-point addition, subtraction, and multiplication operations.

Machine Zero For 64-bit operands, the value of #8000 0000 0000 0000. For 32-bit operands, the value of #8000 0000.

Monitor Interval Timer A 32-bit timer set during an exchange to job mode that sends an interrupt during a hang or endless loop (or similar condition) that allows monitor to regain control of the processor.

Monitor Mode The period in central processor operations during which monitor software and other system processes perform system work; user processes do not execute in monitor mode.

Normalized result An operation that shifts a floating-point result's coefficient left one bit and decreases the exponent by one until the sign bit and the bit immediately to its right are different.

Order Vector A bit vector. Used to determine the positional significance of elements of a vector. Enables the original vector to be regenerated.

Page An allocation unit of CP memory.

Page Fault A page fault occurs when a process requests a page not currently in central processor memory.

Page Table A table that contains the page table entries. The hardware uses a copy of the page table to associate a virtual address with a physical address.

Physical Addressing A addressing scheme that provides the means for a CPU's monitor code to address all of CP memory with a 48-bit bit address. Any physical address beyond the maximum CP memory address will wrap around within CP memory.

Qualifier A mnemonic coded on an instruction line; it controls the setting of one or more G-bits in the instruction's subfunction field.

Register File A set of 256 directly addressed, 64-bit general purpose registers in the central processor. The lower 128 64-bit registers can also be addressed as 256 32-bit registers. Scalar instructions reference its registers as locations of source and result operands. Vector instructions reference registers containing memory locations of source and result operands.

Right Normalization An operation that shifts a floating-point result right one place with sign extension, adding one to the exponent. Performed when a result coefficient overflows its register.

Semaphore A structure that provides facilities to synchronize and pass information in the communication buffer between parallel programs running on the ETA10. Consists of two inter-related sets of words in the communication buffer.

Shared memory Memory accessible from all central processors in the system for bi-directional transfers of information in blocks of arbitrary length from any half word or full word storage unit.

Sign control An operation performed on the input operands of certain instructions. The type of operation is determined by G-bits 5, 6, and 7 in the instruction.

Significant result The number of bit positions in the coefficient excluding the sign bit, minus the left shift count necessary to normalize that number for floating-point operations.

Sparse Vectors Vectors with a great many zero or near-zero elements. Special instructions reorder such vectors to minimize the storage and calculation of (near) zero elements, while maintaining their positional significance.

Trace Register The 64-bit register 00. Its contents are swapped with the contents of the appropriate CP memory location for register 00 during an exchange operation.

Transfer Request Block A four-word block of information used in shared memory data transfers, containing information about the transfer,

Upper result The left half of a result coefficient generated by floating-point addition, subtraction, and multiplication operations.

Virtual Addressing An addressing scheme that provides the means for a CPU's job code to address all of a job's storage, both inside and outside CP memory, with a 48-bit address.

Referenced Documents List

FROM ETA SYSTEMS:

Number	Title
PUB–1005	ETA10 System Reference Manual: [EOS Version 1]
PUB–1050	CDC CYBER 200 Assembler Version 2 Reference Manual (META book)
PUB–1257	ETA/SV Programmers Reference Manual (for "as")
PUB–1255	FORTRAN 77 Reference Manual
PUB–1267	CYBIL Reference Manual

Index

A

A designator, with vector operands, H-1

Absolute, transmit
  #59 instruction, 3-72
  #79 instruction, 3-93

Absolute value, move, instruction #99, 3-123

Add Address
  #63 instruction, 3-78
  #83 instruction, 3-101

Add to Length Field, #2B instruction, 3-38

Add; Lower Result
  #41 instruction, 3-57
  #61 instruction, 3-77
  #81 instruction, 3-99
  #A1 instruction, 3-132

Add; Normalized Result
  #42 instruction, 3-57
  #62 instruction, 3-78
  #82 instruction, 3-100
  #A2 instruction, 3-134

Add; Upper Result
  #40 instruction, 3-56
  #60 instruction, 3-77
  #80 instruction, 3-98
  #A0 instruction, 3-130

Addition, floating-point, F-6

Address
  add
    instruction #63, 3-78
    instruction #83, 3-101
  branch to immediate, instruction #B6, 3-188
  subtract, 3-80
    instruction #87, 3-105

Adjacent Mean, instruction #D1, 3-213

Adjust Exponent
  #55 instruction, 3-68
  #75 instruction, 3-90
  #95 instruction, 3-118

Adjust Significance
  #54 instruction, 3-67
  #74 instruction, 3-89
  #94 instruction, 3-116

AND, logical, instruction #F1, 3-225

AND logical operation, #9D instruction, 3-128

AND NOT, logical, instruction #F6, 3-230

AND NOT logical operation, #9D instruction, 3-128

Arithmetic Compress, instruction #CF, 3-211

Associative Registers
  Load, #0D instruction, 3-11
  Store, #0C instruction, 3-11

Average, instruction #D0, 3-212

Average Difference, instruction #D4, 3-214

a qualifier, 2-13, 2-14, 2-15, 3-70, 3-98, 3-99, 3-100, 3-101, 3-102, 3-103, 3-104, 3-105, 3-106, 3-107, 3-108, 3-109, 3-110, 3-111, 3-112, 3-113, 3-114, 3-115, 3-116, 3-118, 3-120, 3-121, 3-122, 3-123, 3-124, 3-125, 3-126, 3-129, 3-131, 3-133, 3-135, 3-137, 3-139, 3-141, 3-143, 3-145, 3-147, 3-149, 3-151, 3-193, 3-195, 3-197, 3-201, 3-202, 3-203, 3-204, 3-212, 3-214, 3-222

B

B designator, with vector operands, H-1

Backward Domain Change, #17 instruction, 2-5, 3-24

Base/Limit/Access Pair, 2-3

BCD to Binary conversion, #10 instruction, 3-16

Binary compare, masked, instruction #CC, 3-209

Binary to BCD conversion, #11 instruction, 3-17

Bit assignments, data flag register, G-4

Bit Branch and Alter
  #32 instruction, 3-44
  Data Flag Register, #33 instruction, 3-46
  Register Bit Branch and Alter, #2F instruction, 3-41

Bit Branch and Load/Store, instruction #FD, 3-238

Bit Branch and Swap, instruction #FC, 3-237

Bit Compress, #14 instruction, 3-19

Bit Mask, #16 instruction, 3-22

Bit Merge, #15 instruction, 3-20

Bit mask, form repeated
with leading ones, 3-27
with leading zeros, 3-27

BLAP. *See* Base/Limit/Access Pair

Branch
after decrement, instruction #35, 3-49
after increment, instruction #31, 3-43

Branch and load/store, bit, instruction #FD, 3-238

Branch and swap, bit, instruction #FC, 3-237

Branch if Equal
#20 instruction, 3-29
#24 instruction, 3-33

Branch if equal
compare floating-point, instruction #B0, 3-176
compare integers, instruction #B0, 3-152

Branch if Greater or Equal
#22 instruction, 3-31
#26 instruction, 3-35
compare integers, instruction #B2, 3-156

Branch if greater
compare floating-point, instruction #B5, 3-181
compare integers, instruction #B5, 3-162

Branch if greater or equal, compare floating-point, instruction #B2, 3-178

Branch if Less
#23 instruction, 3-32
#27 instruction, 3-36

Branch if less, compare integers, instruction #B3, 3-158

Branch if less or equal
compare floating-point, instruction #B4, 3-180
compare intergers, instruction #B4, 3-160

Branch if less than, compare floating-point, instruction #B3, 3-179

Branch if Not Equal
#21 instruction, 3-30
#25 instruction, 3-34
compare integers, instruction B1, 3-154

Branch if not equal, compare floating-point, instruction #B1, 3-177

Branch or Forward Domain Change, #36 instruction, 2-5, 3-50

Branch to Immediate Address, instruction #B6, 3-188

Branching and indexing, 2-5

Breakpoint on Address, #04 instruction, 3-4

Breakpoint register, 3-4

Broadcast
instructions, D-1
with vector operations, 2-15

b qualifier, 2-13, 2-14, 2-15, 3-70, 3-98, 3-99, 3-100, 3-101, 3-102, 3-103, 3-104, 3-105, 3-106, 3-107, 3-108, 3-109, 3-110, 3-111, 3-116, 3-118, 3-125, 3-129, 3-131, 3-133, 3-135, 3-137, 3-139, 3-141, 3-143, 3-145, 3-147, 3-149, 3-151, 3-189, 3-193, 3-195, 3-197, 3-201, 3-202, 3-203, 3-204, 3-211, 3-212, 3-214, 3-222

br qualifier, 2-13, 3-41, 3-44, 3-46

brb qualifier, 2-13, 3-41

brf qualifier, 2-13, 3-41

bro qualifier, 2-13, 3-41, 3-44, 3-46

brz qualifier, 2-13, 3-41, 3-44, 3-46

C

C + 1 designator, 2-15

C designator, 2-15

Ceiling
#52 instruction, 3-65
#72 instruction, 3-87
#92 instruction, 3-114

Central processing unit, 2-2

Communication buffer, 2-3

Compare, masked binary, instruction #CC, 3-209

Compare Floating-Point, Branch if Equal, instruction #B0, 3-176

Compare Floating-Point, Branch if Greater, instruction #B5, 3-181

Compare Floating-Point, Branch if Greater or Equal, instruction #B2, 3-178

Compare Floating-Point, Branch if Less or Equal, instruction #B4, 3-180

Compare Floating-Point, Branch if Less Than, instruction #B3, 3-179

Compare Floating-Point, Branch if Not Equal, instruction #B1, 3-177

Compare Floating-Point, Set Condition if Equal, instruction #B0, 3-182

Compare Floating-Point, Set Condition if Greater or Equal, instruction #B2, 3-184

Compare Floating-Point, Set Condition if Less or Equal, instruction #B4, 3-186

Compare Floating-Point, Set Condition if Less Than, instruction #B3, 3-185

Compare Floating-Point, Set Condition if Not Equal, instruction #B1, 3-183

Compare Floating-Point, Set Condtion if Greater, instruction #B5, 3-187

Compare Integers, Branch if Equal, instruction #B0, 3-152

Compare Integers, Branch if Greater, instruction #B5, 3-162

Compare Integers, Branch if Less, instruction #B3, 3-158

Compare Integers, Branch if Not Equal, instruction #B1, 3-154

Compare Integers, Set Condition if Equal, instruction #B0, 3-164

Compare Integers, Set Condition if Greater, instruction #B5, 3-174

Compare Integers, Set Condition if Greater or Equal, instruction #B2, 3-168

Compare Integers, Set Condition if Less, instruction #B3, 3-170

Compare Integers, Set Condition if Less or Equal, instruction #B4, 3-172

Compare Integers, Set Condition if Not Equal, instruction #B1, 3-166

Compare Integers; Branch if Greater or Equal, instruction #B2, 3-156

Compare Integers; Branch if Less or Equal, instruction #B4, 3-160

Compare, Greater Than or Equal, instruction #C6, 3-203

Compare; Equal, instruction #C4, 3-201

Compare; Less, instruction #C7, 3-204

Compare; Not Equal, instruction #C5, 3-202

Comparison rules, floating-point, F-18

Compress
- arithmetic, instruction #CF, 3-211
- instruction #BC, 3-194

Compress Bits, #14 instruction, 3-19

Contract
- #76 instruction, 3-91
- #96 instruction, 3-120
- rounded
  - instruction #77, 3-92
  - instruction #97, 3-121

Control vector, 2-14, 3-98, 3-99, 3-100, 3-101, 3-102, 3-103, 3-104, 3-105, 3-106, 3-107, 3-108, 3-109, 3-110, 3-111, 3-112, 3-113, 3-114, 3-115, 3-116, 3-118, 3-120, 3-121, 3-122, 3-123, 3-124, 3-125, 3-126, 3-129

Convert BCD to Binary, #10 instruction, 3-16

Convert Binary to BCD, #11 instruction, 3-17

Count Leading Equals, #1E instruction, 3-28

Count Leading Ones, #1F instruction, 3-28

CP Memory
- load byte instruction, 3-18
- store byte instruction, 3-18

CP memory, 2-3

CPU. *See* Central processing unit

CPU cycle counter. *See* Instrumentation counter

CPU instructions, execution mode, 3-7

c qualifier, 2-13, 2-14, 2-15, 3-98, 3-99, 3-100, 3-102, 3-103, 3-104, 3-106, 3-107, 3-109, 3-110, 3-111, 3-115, 3-131, 3-133, 3-135, 3-137, 3-139, 3-141, 3-143, 3-145, 3-147, 3-149, 3-151, 3-211

ca0 qualifier, 2-13

ca1 qualifier, 2-13

ca2 qualifier, 2-13

ca3 qualifier, 2-13

D

Data Flag Bits, set by function codes, G-7

Data Flag Register
- Data Flag Register Bit Branch and Alter, #33 instruction, 3-46
- Load/Store, #3B instruction, 3-54

Data flag branch, G-3

Data flag branch enable bit, in data flag register, G-2

Data flag branch register
- data flags, G-4, G-5
- mask field, G-4, G-5
- product field, G-4, G-5

Data flag register, G-1
- bit assignments, G-4
- data flag branch enable bit, G-2
- data flags, G-2
- format, G-1
- free data flags, G-5
- mask field, G-2
- product field, G-2

Data flags
in data flag branch register, G-4, G-5
in data flag register, G-2

Decrease (R) and Branch, #35 instruction, 3-49

Delta, instruction #D5, 3-215

Designators, instruction, 2-6, 2-12

Difference, average, instruction #D4, 3-214

Divide; Significant Result
#4F instruction, 3-62
#6F instruction, 3-84
#8F instruction, 3-111
instruction #AF, 3-150

Divide; Upper Result
#4C instruction, 3-61
#6C instruction, 3-82
#8C instruction, 3-110
instruction #AC, 3-148

Division, floating-point, F-12

Domain Change
backward, #17 instruction, 3-24
Forward, #36 instruction, 3-50

Domain registers, reading, #57 instruction, 3-71

Dot Products of Vectors A and B, instruction #DC, 3-222

Double-precision result, floating-point, F-15

data flag bits set by instructions, G-6

E

Enter (R) With I (48 Bits), instruction #BE, 3-196

Enter (R) with I, #3E instruction, 3-55

Enter Length of (R) with (I), 3-38

Equal Byte, scan for, #28 instruction, 3-37

Equality, search for, instruction #C8, 3-205

ETA10
hardware, 1-2, 2-2
instruction formats, 1-3
instruction functions, 1-3
instruction operands, 1-3
instruction operations, 1-2
memories, 2-3

Exchange
Job to Monitor Mode, #09 instruction, 3-9
Monitor to Job Mode, #09 instruction, 3-9

Exclusive OR, logical, instruction #F0, 3-224

Exclusive OR logical operation, #9D instruction, 3-128

Exclusive OR NOT, logical, instruction #F7, 3-231

Exclusive OR NOT logical operation, #9D instruction, 3-128

Excusive OR, logical, instruction #2C, 3-39

Execution mode, instruction, 3-7

Exit Force, #09 instruction, 2-2, 2-5, 3-9

Exponent
adjust
instruction #55, 3-68
instruction #95, 3-118
adjust (64 bits), instruction #75, 3-90
move, instruction #9A, 3-124
transmit
#5A instruction, 3-73
#7A instruction, 3-94

Extend
#5C instruction, 3-74
#9C instruction, 3-126
index, instruction #5D, 3-75

External interrupt, 3-13
transmit, 3-8

Extract Bits, #6E instruction, 3-83

F

Fault Test, #06 instruction, 3-6

Floating-point arithmetic, two's complement, F-9

Floating-point compare, set conditon if greater, instruction #B5, 3-187

Floating-point compare, brach if equal, instruction #B0, 3-176

Floating-point compare, branch if greater, instruction #B5, 3-181

Floating-point compare, branch if greater or equal, instruction #B2, 3-178

Floating-point compare, branch if less or equal, instruction #B4, 3-180

Floating-point compare, branch if less than, instruction #B3, 3-179

Floating-point compare, branch if not equal, instruction #B1, 3-177

Floating-point compare, set condition if equal, instruction #B0, 3-182

Floating-point compare, set condition if greater or equal, instruction #B2, 3-184

Floating-point compare, set condition if less or equal, instruction #B4, 3-186

Floating-point compare, set condition if less than, instruction #B3, 3-185

Floating-point compare, set condition if not equal, instruction #B1, 3-183

Floating-point
addition, 3-98, 3-99, 3-100, 3-130, 3-132, 3-134
division, 3-110, 3-111, 3-148, 3-150
multiplication, 3-106, 3-107, 3-109, 3-142, 3-144, 3-146
operations, 2-5
subtraction, 3-102, 3-103, 3-104, 3-136, 3-138, 3-140

Floating-point format
32-bit, F-1
64-bit, F-1

Floating-point operations, F-1
Addition, F-6
comparison rules, F-18
division, F-12
double-precision result, F-15
lower result, F-4
Multiplication, F-10
normalization, F-5
Subtraction, F-8
significant result, F-17
square root, F-16
upper result, F-4

Floor
#51 instruction, 3-64
#71 instruction, 3-86
#91 instruction, 3-113

Formats for instructions, 2-6

Forward Domain Change, #36 instruction, 3-50

Free data flags, in data flag register, G-5

Function codes, setting Data Flag bits, G-7

fia qualifier, 2-13

fwc qualifier, 2-13

G

G-bit, 2-12, 2-14
sign control, C-1

G-field, 2-12, 2-14

Gather, instruction #BA, 3-192

Greater, search for, instruction #CA, 3-207

grp qualifier, 2-13

H

Half Word Enter (R) by I (24 Bits), instruction #CD, 3-210

Half Word Enter (R) with I, #4D instruction, 3-61

Half Word Increase (R) by I, #4E instruction, 3-62

Half Word Increase (R) by I (24 Bits), instruction #CE, 3-210

Half Word Index Multiply, #3C instruction, 3-54

Halfword
load, instruction #5E, 3-76
store, instruction #5F, 3-76

h qualifier, 2-13, 2-14, 2-15, 3-69, 3-98, 3-99, 3-100, 3-102, 3-103, 3-104, 3-106, 3-107, 3-109, 3-110, 3-111, 3-112, 3-113, 3-114, 3-115, 3-116, 3-118, 3-122, 3-123, 3-124, 3-125, 3-128, 3-130, 3-132, 3-134, 3-136, 3-138, 3-140, 3-142, 3-144, 3-146, 3-148, 3-150, 3-152, 3-154, 3-156, 3-158, 3-160, 3-162, 3-164, 3-166, 3-168, 3-170, 3-172, 3-174, 3-182, 3-183, 3-184, 3-185, 3-186, 3-187, 3-189, 3-191, 3-192, 3-193, 3-194, 3-195, 3-197, 3-198, 3-199, 3-200, 3-201, 3-202, 3-203, 3-204, 3-205, 3-206, 3-207, 3-208, 3-211, 3-212, 3-213, 3-214, 3-215, 3-216, 3-218, 3-220, 3-221, 3-222, 3-223, 3-237, 3-238, 3-239

I

Idle, #00 instruction, 3-3

Illegal instruction mask, Domain Package, I-2

Inclusive OR, logical
instruction #2E, 3-40
instruction #F2, 3-226

Increase (R) and Branch, #31 instruction, 3-43

Increase (R) By I (48 Bits), instruction #BF, 3-196

Increase (R) by I, #3F instruction, 3-56

Index Extend, #5D instruction, 3-75

Index Multiply, #3D instruction, 3-55

Inequality, search for, instruction #C9, 3-206

Input Queue Valid flag, 3-26

Input/Output Unit, 2-3

Insert Bits, #6D instruction, 3-83

Instruction
broadcast, D-1
description format, 3-2
designators, 2-6, 2-12
floating-point operations, F-1
formats, 2-6
function, 2-12
function field, 2-12
offset, H-1
operations, 2-4
sorted by code, A-1
sorted by mnemonic, B-1
subfunction, 2-12
termination rules, E-1
with sign control, C-1

Instruction qualifiers, table, 2-13

Instructions, illegal, I-1

Instructions affecting data flag register bits, G-6

Instrumentation Counter, transmit, #29 instruction, 3-37

Internal interrupt, 3-13

Interrupt
external, 3-13
transmit, 3-8
internal, 3-13

Interrupt Register, read, #0E instruction, 3-12

Interrupt register, 3-12

Interval, instruction #DF, 3-223

Interval timer, Monitor, 3-10

IOU. *See* Input/Output Unit

illegal instructions, I-1

ivg qualifier, 2-13, 3-130, 3-132, 3-134, 3-136, 3-138, 3-140, 3-142, 3-144, 3-146, 3-148, 3-150

J

Job Interval Timer
transmit, instruction #37, 3-52
transmit to, instruction #3A, 3-53

Job mode, 2-2

Job to Monitor Mode exchange, #09 instruction, 3-9

K

Keys, loading, #0F instruction, 3-14

L

Leading Equals, count, #1E instruction, 3-28

Leading Ones, count, #1F instruction, 3-28

Length, transmit, #7C instruction, 3-95

Length Field, add to, #2B instruction, 3-38

Less, search for, instruction #CB, 3-208

Link, select, instruction #56, 3-69

Linked vector instructions, valid combinations, 3-70

Load
#5E instruction, 3-76
#7E instruction, 3-97

Load Associative Registers, #0D instruction, 3-11

Load Byte, #12 instruction, 3-18

Load Data Flag Register, #3B instruction, 3-54

Load Keys, #0F instruction, 3-14

Load Register, instruction #FE, 3-239

Load/Store, bit branch and, instruction #FD, 3-238

Logical AND
#2D instruction, 3-39
instruction #F1, 3-225

Logical AND NOT, instruction #F6, 3-230

Logical Exclusive OR
#2C instruction, 3-39
instruction #F0, 3-224

Logical Exclusive OR NOT, instruction #F7, 3-231

Logical Inclusive OR
#2E instruction, 3-40
instruction #F2, 3-226

Logical NOT AND, instruction #F3, 3-227

Logical NOT OR, instruction #F4, 3-228

Logical Operation, #9D instruction, 3-128

Logical OR NOT, instruction #F5, 3-229

Lower result
add
instruction #41, 3-57
instruction #61, 3-77
instruction #81, 3-99
instruction #A1, 3-132
multiply
instruction #49, 3-60
instruction #69, 3-81
instruction #89, 3-107

instruction A9, 3-144
subtract
instruction #45, 3-58
instruction #65, 3-79
instruction #85, 3-103
instruction #A5, 3-138

lh qualifier, 2-13

M

Mask, instruction #BB, 3-193

Mask Bits, #16 instruction, 3-22

Mask field
in data flag branch register, G-4, G-5
in data flag register, G-2

Masked Binary Compare, instruction #CC, 3-209

Maximum of Vector A, instruction #DA, 3-216

Memory operations, 2-4

Memory transfer, shared memory, instruction #18, 3-24

Merge, instruction #BD, 3-195

Merge Bits, #15 instruction, 3-20

Minimum of Vector A, instruction #D9, 3-218

Mode
Job, 2-2
Monitor, 2-2, 2-5

Monitor mode, 2-2, 2-5

Monitor to Job Mode exchange, #09 instruction, 3-9

Move Absolute, #99 instruction, 3-123

Move Bytes Left, instruction #F8, 3-232

Move Exponent, #9A instruction, 3-124

Multiplication, floating-point, F-10

Multiply
half word index, instruction #3C, 3-54
Index, instruction #3D, 3-55

Multiply; Lower Result
#49 instruction, 3-60
#69 instruction, 3-81
#89 instruction, 3-107
instruction A9, 3-144

Multiply; Significant Result
#4B instruction, 3-60
#6B instruction, 3-82
#8B instruction, 3-109
instruction #AB, 3-146

Multiply; Upper Result
#48 instruction, 3-59
#68 instruction, 3-81
#88 instruction, 3-106
instruction #A8, 3-142

ma qualifier, 2-13, 2-14, 2-15, 3-98, 3-99, 3-100, 3-102, 3-103, 3-104, 3-106, 3-107, 3-109, 3-110, 3-111, 3-115, 3-131, 3-133, 3-135, 3-137, 3-139, 3-141, 3-143, 3-145, 3-147, 3-149, 3-151, 3-211, 3-216, 3-218

mb qualifier, 2-13, 2-14, 2-15, 3-98, 3-99, 3-100, 3-102, 3-103, 3-104, 3-106, 3-107, 3-109, 3-110, 3-111, 3-131, 3-133, 3-135, 3-137, 3-139, 3-141, 3-143, 3-145, 3-147, 3-149, 3-151, 3-211

N

NOT AND, logical, instruction #F3, 3-227

NOT AND logical operation, #9D instruction, 3-128

NOT OR, logical, instruction #F4, 3-228

NOT OR logical operation, #9D instruction, 3-128

No Operation, #03 instruction, 3-3

Normalization, floating-point, F-5

Normalized result
add
instruction #42, 3-57
instruction #62, 3-78
instruction #82, 3-100
instruction #A2, 3-134
subtract
instruction #66, 3-80
instruction #86, 3-104
instruction A6, 3-140

Normalized upper results, F-14

n qualifier, 2-13, 2-14, 2-15, 3-98, 3-99, 3-100, 3-102, 3-103, 3-104, 3-106, 3-107, 3-109, 3-110, 3-111, 3-131, 3-133, 3-135, 3-137, 3-139, 3-141, 3-143, 3-145, 3-147, 3-149, 3-151, 3-211

neq qualifier, 2-13

O

Offset
with vector operands, H-1
with vector operations, 2-15

One's complement number, F-2

Operand
shift
#30 instruction, 3-42
#34 instruction, 3-48
transmit
#58 instruction, 3-72
#78 instruction, 3-93

OR logical operation, #9D instruction, 3-128

OR NOT, logical, instruction #F5, 3-229

OR NOT logical operation, #9D instruction, 3-128

Order vector, 3-130, 3-132, 3-134, 3-136, 3-138, 3-140, 3-142, 3-144, 3-146, 3-148, 3-150, 3-193, 3-194, 3-195, 3-201, 3-202, 3-203, 3-204, 3-211

o qualifier, 2-13, 2-14, 2-15, 3-69, 3-98, 3-99, 3-100, 3-101, 3-102, 3-103, 3-104, 3-105, 3-106, 3-107, 3-108, 3-109, 3-110, 3-111, 3-112, 3-113, 3-114, 3-115, 3-116, 3-118, 3-120, 3-121, 3-122, 3-123, 3-124, 3-125, 3-126, 3-191, 3-212, 3-213, 3-214, 3-215

P

Pack
#5B instruction, 3-73
#7B instruction, 3-94
#9B instruction, 3-125

Post Semaphore, instruction #FA, 3-233

Product field
in data flag branch register, G-4, G-5
in data flag register, G-2

Product of Vector A Elements, instruction #DB, 3-221

pa0 qualifier, 2-13

pa1 qualifier, 2-13

pa2 qualifier, 2-13

pa3 qualifier, 2-13

R

Read Domain Registers, #57 instruction, 3-71

Read Interrupt Register, #0E instruction, 3-12

Real Time Clock, transmit, instruction #39, 3-53

Register
load, instruction FE, 3-239
store, instruction #FF, 3-239

Register Bit Branch and Alter, #2F instruction, 3-41

Register R, structure for #0F instruction, 3-15

Repeated Bit Mask
with leading ones, #1D instruction, 3-27
with leading zeros, #1C instruction, 3-27

Result formats, floating-point arithmetic, F-4

Rounded Contract
#77 instruction, 3-92
#97 instruction, 3-121

ra qualifier, 2-13, 3-69

rb qualifier, 2-13, 3-69

rel qualifier, 2-13

rf qualifier, 2-13

rvg qualifier, 2-13, 3-130, 3-132, 3-134, 3-136, 3-138, 3-140, 3-142, 3-144, 3-146, 3-148, 3-150

S

Scalar operations, 2-4

Scan for Equal Byte, #28 instruction, 3-37

Scatter, instruction #B7, 3-189

SECDED circuitry, fault test instruction, 3-6

Search for Equality, instruction #C8, 3-205

Search for Greater, instruction #CA, 3-207

Search for Inequality, instruction #C9, 3-206

Search for Less, instruction #CB, 3-208

Select Equal; A=B, Item Count to (C), instruction #C0, 3-197

Select Greater or Equal; A GE B, Item Count to (C), instruction #C2, 3-199

Select Less; A LT B, Item Count to (C), instruction #C3, 3-200

Select Link, #56 instruction, 3-69

Select Not Equal; A NE B, Item Count to (C), instruction #C1, 3-198

Select Serial/Parallel Execution Mode, #07 instruction, 3-7

Semaphore
post, instruction #FA, 3-233
wait on, instruction #FB, 3-235

Service Unit, 2-3

Set condition if equal
  compare floating-point, instruction #B0, 3-182
  compare integers, instruction #B0, 3-164

Set condition if greater
  compare floating-point, instruction #B5, 3-187
  compare integers, instruction #B5, 3-174

Set condition if greater or equal
  compare floating-point, instruction #B2, 3-184
  compare integers, instruction #B2, 3-168

Set condition if less, compare integer, instruction #B3, 3-170

Set condition if less or equal, compare integers, instruction #B4, 3-172

Set condition if less than, compare floating-point, instruction #B3, 3-185

Set condition if not equal
  compare floating-point, instruction #B1, 3-183
  compare integers, instruction #B1, 3-166

Set condtion if less or equal, compare floating-point, instruction #B4, 3-186

Shared memory, 2-3
  Start I/O, #19 instruction, 3-25
  Stop I/O, #1A instruction, 3-25
  Test I/O, #1B instruction, 3-26

Shared memory transfer, #18 instruction, 3-24

Shift Element, #8A instruction, 3-108

Shift Operand
  #30 instruction, 3-42
  #34 instruction, 3-48

Sign control, 2-15, 3-69, 3-98, 3-99, 3-100, 3-102, 3-103, 3-104, 3-106, 3-107, 3-109, 3-110, 3-111, 3-115, 3-131, 3-133, 3-135, 3-137, 3-139, 3-141, 3-143, 3-145, 3-147, 3-149, 3-151, 3-211, 3-216, 3-218, C-1

Significance, adjust
  instruction #54, 3-67
  instruction #74, 3-89
  instruction #94, 3-116

Significant result
  divide
    instruction #4F, 3-62
    instruction #8F, 3-111
    instruction #AF, 3-150
  floating-point, F-17
  multiply
    instruction #4B, 3-60
    instruction #8B, 3-109
    instruction #AB, 3-146

Significant results
  divide, instruction #6F, 3-84
  multiply, instruction #6B, 3-82

Significant Square Root
  #53 instruction, 3-66
  #73 instruction, 3-88
  #93 instruction, 3-115

Sparse vector, 3-130, 3-132, 3-134, 3-136, 3-138, 3-140, 3-142, 3-144, 3-146, 3-148, 3-150, 3-194, 3-211

Special purpose registers, access to, 2-5

Square root
  floating-point, F-16
  significant
    instruction #53, 3-66
    instruction #73, 3-88
    instruction #93, 3-115

Start I/O, Shared memory, #19 instruction, 3-25

Stop I/O, Shared memory, #1A instruction, 3-25

Store
  #5F instruction, 3-76
  #7F instruction, 3-97

Store Associative Registers, #0C instruction, 3-11

Store Byte, #13 instruction, 3-18

Store Data Flag Register, #3B instruction, 3-54

Store register, instruction #FF, 3-239

Store/load, bit branch and, instruction #FD, 3-238

Subfunction
  instruction, 2-12
  with vector operations, 2-14

Subtract, normalized result, instruction #46, 3-59

Subtract Address
  #67 instruction, 3-80
  #87 instruction, 3-105

Subtract; Lower Result
  #45 instruction, 3-58
  #65 instruction, 3-79
  #85 instruction, 3-103
  instruction #A5, 3-138

Subtract; Normalized Result
  #46 instruction, 3-59

#66 instruction, 3-80
#86 instruction, 3-104
instruction #A6, 3-140

Subtract; Upper Result
#44 instruction, 3-58
#64 instruction, 3-79
#84 instruction, 3-102
instruction #A4, 3-136

Subtraction, floating-point, F-8

Sum Vector A Elements, instruction DA, 3-220

Swap, #7D instruction, 3-96

sa0 qualifier, 2-13

sa1 qualifier, 2-13

sa2 qualifier, 2-13

sa3 qualifier, 2-13

sb qualifier, 2-13

sc qualifier, 2-13

so qualifier, 2-13, 3-41, 3-44, 3-46

sz qualifier, 2-13, 3-41, 3-44, 3-46

T

Termination rules for instructions, E-1

Test I/O, Shared memory, #1B instruction, 3-26

TRB. *See* Transfer Request Block

Trace register, use by Idle instruction, 3-3

Transfer Request Block, 2-4

Transmit (R) to (T), #38 instruction, 3-52

Transmit (R) to Job Interval Timer, #3A instruction, 3-53

Transmit (R) to Monitor Interval Timer, #0A instruction, 3-10

Transmit Absolute
#59 instruction, 3-72
#79 instruction, 3-93

Transmit Element, #98 instruction, 3-122

Transmit Exponent
#5A instruction, 3-73
#7A instruction, 3-94

Transmit External Interrupt, #08 instruction, 3-8

Transmit Instrumentation Counter, #29 instruction, 3-37

Transmit Job Interval Timer to (T), #37 instruction, 3-52

Transmit Length, #7C instruction, 3-95

Transmit Operand
#58 instruction, 3-72
#78 instruction, 3-93

Transmit Real-Time Clock to (T), #39 instruction, 3-53

Transmit Reverse, instruction #B8, 3-191

Truncate
#50 instruction, 3-63
#70 instruction, 3-85
#90 instruction, 3-112

Two's complement
conversion to, 3-16
convert from, to BCD, 3-17

Two's complement number, F-2

Type One illegal instructions, I-1

Type Two illegal instructions, I-1

t qualifier, 2-13, 3-41, 3-44, 3-46

U

Upper result
add
instruction #40, 3-56
instruction #60, 3-77
instruction #80, 3-98
instruction #A0, 3-130
divide, 3-110
instruction #4C, 3-61
instruction #6C, 3-82
instruction #AC, 3-148
multiply
instruction #48, 3-59
instruction #68, 3-81
instruction #88, 3-106
instruction #A8, 3-142
subtract
instruction #44, 3-58
instruction #64, 3-79
instruction #84, 3-102
instruction #A4, 3-136

usi qualifier, 2-13

V

Vector, shift element, instruction #8A, 3-108

Vector A
maximum of, instruction #D8, 3-216
minimum of, instruction #D9, 3-218

Vector A elements
product of, instruction #DB, 3-221

sum, instruction #DA, 3-220

Vector instructions, used in link operation, 3-69

Vector offset, 2-15

Vector operands, addressing, H-1

Vector operations, 2-4
subfunction, 2-14

Vectors, logical operations, instruction #9D, 3-128

Vectors A and B, dot products of, instruction #DC, 3-222

Void Stack and Branch, #05 instruction, 3-6

W

Wait on Semaphore, instruction #FB, 3-235

Word
load, instruction #7E, 3-97
store, instruction #7F, 3-97

X

xvg qualifier, 2-13, 3-130, 3-132, 3-134, 3-136, 3-138, 3-140, 3-142, 3-144, 3-146, 3-148, 3-150

Z

Z designator, with control vector, 2-14

z qualifier, 2-13, 2-14, 3-69, 3-98, 3-99, 3-100, 3-101, 3-102, 3-103, 3-104, 3-105, 3-106, 3-107, 3-108, 3-109, 3-110, 3-111, 3-112, 3-113, 3-114, 3-115, 3-116, 3-118, 3-120, 3-121, 3-122, 3-123, 3-124, 3-125, 3-126, 3-129, 3-191, 3-194, 3-197, 3-198, 3-199, 3-200, 3-212, 3-213, 3-214, 3-215, 3-216, 3-218, 3-220, 3-221, 3-222, 3-223

Reader Comment Sheet

Providing our readers with effective documentation is one of our most important goals. You can help us improve our documentation by taking a few moments to review this ETA Systems publication. If you fill out this comment sheet and include your name and address on the reverse side, we will send you an ETA Systems pen to thank you for your help.

☐ Yes ☐ Somewhat ☐ No Is this publication easy to read and use?
Comments:

☐ Yes ☐ Somewhat ☐ No Does it tell you what you need to know?
Comments:

☐ Yes ☐ Somewhat ☐ No Is the organization of topics logical?
Comments:

☐ Yes ☐ Somewhat ☐ No Are there enough examples?
Comments:

☐ Yes ☐ Somewhat ☐ No Are the examples helpful?
Comments:

☐ Yes ☐ Somewhat ☐ No Are the illustrations effective?
Comments:

- Are there any errors in this publication? (Please list errors in the format shown below if possible.)
  Page number *Description of Error*

Please include your name and address so we can send you an ETA Systems pen. If you would like a reply to any questions about the document, check off the applicable box. Fold this sheet on the dotted lines, seal it with tape, and send it to the address below. Thank you for your time and input.

☐ Yes, I would like a reply. ☐ No, I don't want a reply.

Date:

Company: Phone:

Street Address:

City: State: Zip/Country:

fold 1

fold 2

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 2294 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

ETA SYSTEMS, INCORPORATED
TECHNICAL COMMUNICATION DEPT.
1450 ENERGY PARK DRIVE
ST. PAUL, MN 55108